Progress and Preservation

CAS CS 320: Concepts of Programming Languages

November 3, 2026

We've come to what's probably the most important topic in this course. All those times we've said we want to design "good" PLs, we can now say what that means. The kind of "goodness" we're interested in (at the moment) is *theoretical* goodness. We'll be ignoring "softer" concerns like ergonomics, expressibility, tooling, etc.

Theoretical goodness comes down to the following basic concern: as we've said, a PL is specified by a (1) syntax, (2) type system, and (3) semantics. We know what makes a syntax (theoretically) good: it should be unambiguous, i.e., there's only one way to interpret the structure of a program written in the language. *But what makes a type system and a semantics (theoretically) good?* Again, we're ignoring soft concerns like "hey, maybe don't use the `+` symbol to mean integer multiplication (or worse, string concatenation), that'd be confusing." Or "maybe don't make `int` the expected type of an if-expression condition." For now, we just say: "you're the PL designer, it's your prerogative."

But, after the dust settles, and you have a type system and a semantics, *they better make sense together*. We express this desired coherence of type system and semantics in two ways:

- well-typed programs that aren't "done" being evaluated can continue to be evaluated; this is called ***progress***;
- well-typed programs don't *change* their type (or become ill-typed) during evaluation; this is called ***preservation***, or sometimes ***subject reduction***.

And that's it; a good PL is one that satisfies progress and preservation.[1] All that's left is to turn it into math.

Progress and Preservation. Suppose we've defined a PL with abstract syntax for expressions (e) and types (τ), a type system over judgments of the form $\Gamma \vdash e : \tau$ (where Γ is a context) and a small-step semantics $e_1 \longrightarrow e_2$ with a fixed notion of value (v). ***Progress*** is the property that, for any expression e, if $\varnothing \vdash e : \tau$ for some type τ then e is either a value, or there's an expression e' such that $e \longrightarrow e'$. Recall that an expression is called ***stuck*** if it's irreducible but not a value. In a PL that satisfies progress, *well-typed expressions can't*

[1] Of course, that's not it, even theoretically, but it makes for a more satisfying pedagogical storyline to say that it is.

get stuck; if an expression isn't already a value, then we can "make progress" towards one.

One upshot is that, theoretically speaking, we don't need to deal with runtime errors. This, of course, isn't true in practice—there are more reasons for runtime errors than stuck terms (e.g., IO errors)—but it substantially simplifies the evaluator, both in terms of implementation and reasoning. It's also just pretty, mathematically speaking. Typing is a local affair; it deals with the well-defined composition of constituent parts. But (multi-step) evaluation is global; complex dynamics govern the structure of an expression during evaluation. Progress tells us we can know exactly what possible shapes an expression will be in after evaluation, no matter where it started, or how it looks intermediately.

Preservation is the property that, if $\Gamma \vdash e : \tau$ and $e \longrightarrow e'$ then $\Gamma \vdash e' : \tau$. In PLs that satisfy preservation, well-typed expressions have the same type before and after (single-step) evaluation. Together with progress, this gives us even more information about how things look after evaluation. For example, we can use progress and preservation in languages with Booleans to show that if $\varnothing \vdash e : \mathsf{bool}$, and e evaluates to a value v, then $v = \mathsf{true}$ or $v = \mathsf{false}$. These are called ***canonical values***; given progress and preservation, we can show well-typed expressions evaluate to canonical values.

Induction on derivations. To demonstrate a PL satisfies progress and/or preservation we need to be able to reason about well-typed expressions. We do this by proxy, by reasoning about typing derivations; the well-typed expressions are exactly the expressions that appear as subjects at the roots of typing derivations. Typing derivations are trees, which are inductively defined structures, so naturally we'll use induction to reason about them.

In general, induction is a principle used to prove universal statements about inductively defined structures, e.g. natural numbers, trees, and (you guessed it) derivations. If you've taken a course in discrete mathematics, you've likely seen the principle of natural number induction:

Given a property P of natural numbers, if:

- *P holds of the number 0;*
- *for any number k, if we assume P holds of k (an assumption often called the **induction hypothesis**), then we can show that P holds of $k + 1$;*

then P holds of every natural number.

Inductive structures—by definition of being inductive—have analogous induction principles. Here's the principle of induction for derivations:

Given a property P of judgments, if:

- *for any judgment J and derivable judgments $J_1, \ldots, J_k$, if we assume that P holds of each judgment $J_1, \ldots, J_k$ where*

$$\frac{}{\mathcal{C} \longrightarrow^\star \mathcal{C}}\ \textsc{refl} \qquad\qquad \frac{\mathcal{C}_1 \longrightarrow \mathcal{C}_2 \qquad \mathcal{C}_2 \longrightarrow^\star \mathcal{C}_3}{\mathcal{C}_1 \longrightarrow^\star \mathcal{C}_3}\ \textsc{step}$$

Figure 1: Multi-step semantic rules

$$\frac{J_1 \qquad \ldots \qquad J_k}{J}$$

then we can show that P holds of J;

then P holds of all derivable judgments.

This principle tells us that we can look at the *last* inference rule applied and show that the property holds assuming it holds of the *antecedents* of the last applied rule.

It may be easiest to see how this works by example. Let's start with a simple one about multi-step semantics (Figure 1). The STEP rule allows us to extend a multi-step derivation by one step. *What if we want to extend by multiple steps?*

Proposition 1. *If $e_1 \longrightarrow^\star e_2$ and $e_2 \longrightarrow^\star e_3$, then $e_1 \longrightarrow^\star e_3$.*

Proof. We can prove this by induction on the derivation $\mathfrak{D}$ of $e_1 \longrightarrow^\star e_2$. Let $\mathfrak{D}_2$ denote the derivation of $e_2 \longrightarrow^\star e_3$. This means looking at the different ways the derivation could "end." There are two multi-step rules, so there are two cases to consider.

Case 1. (REFL) Suppose $\mathfrak{D}$ ends with

$$\frac{}{e_1 \longrightarrow^\star e_1}\ \textsc{refl}$$

In particular, that's the *entirety* of $\mathfrak{D}$, and $e_1 = e_2$. This means $\mathfrak{D}_2$ derives $e_1 \longrightarrow^\star e_3$.

Case 2. (STEP) Suppose $\mathfrak{D}$ ends with

$$\frac{\begin{array}{cc}[\mathfrak{D}_1] & [\mathfrak{D}'] \\ e_1 \longrightarrow e & e \longrightarrow^\star e_2\end{array}}{e_1 \longrightarrow^\star e_2}\ \textsc{step}$$

By the induction hypothesis applied to $\mathfrak{D}'$ and $\mathfrak{D}_2$, there is a derivation $\mathfrak{D}_3$ of $e \longrightarrow^\star e_3$. We can then combine $\mathfrak{D}_1$ and $\mathfrak{D}_3$ using the STEP rule to derive $e_1 \longrightarrow^\star e_3$:

$$\frac{\begin{array}{cc}[\mathfrak{D}_1] & [\mathfrak{D}_3] \\ e_1 \longrightarrow e & e \longrightarrow^\star e_3\end{array}}{e_1 \longrightarrow^\star e_3}\ \textsc{step}$$

□

$$e \ ::= \ e + e \mid e - e \mid e * e \mid n$$

Figure 2: Abstract syntax for arithmetic expressions

$$\frac{e_1 \Downarrow v_1 \qquad e_2 \Downarrow v_2}{e_1 + e_2 \Downarrow v_1 + v_2}\ \textsc{add} \qquad \frac{e_1 \Downarrow v_1 \qquad e_2 \Downarrow v_2}{e_1 - e_2 \Downarrow v_1 - v_2}\ \textsc{sub}$$

$$\frac{e_1 \Downarrow v_1 \qquad e_2 \Downarrow v_2}{e_1 * e_2 \Downarrow v_1 \times v_2}\ \textsc{mul} \qquad \frac{n \in \mathbb{Z}}{n \Downarrow n}\ \textsc{intLit}$$

Figure 3: A big-step semantics for arithmetic

Operationally speaking, this proof just composes several STEP rules. For example, given

$$\dfrac{\dfrac{\vdots}{e_1 \longrightarrow e} \qquad \dfrac{\dfrac{\vdots}{e \longrightarrow e'} \qquad \dfrac{\dfrac{\vdots}{e' \longrightarrow e_2} \qquad \dfrac{}{e_2 \longrightarrow^\star e_2}\ \textsc{refl}}{e' \longrightarrow^\star e_2}\ \textsc{step}}{e \longrightarrow^\star e_2}\ \textsc{step}}{e_1 \longrightarrow^\star e_2}\ \textsc{step}$$

and a derivation $\mathfrak{D}$ of $e_2 \longrightarrow^\star e_3$, we can combine them using the procedure in the proof above to show $e_1 \longrightarrow^\star e_3$:

$$\dfrac{\dfrac{\vdots}{e_1 \longrightarrow e} \qquad \dfrac{\dfrac{\vdots}{e \longrightarrow e'} \qquad \dfrac{\dfrac{\vdots}{e' \longrightarrow e_2} \qquad [\mathfrak{D}]}{e' \longrightarrow^\star e_3}\ \textsc{step}}{e \longrightarrow^\star e_3}\ \textsc{step}}{e_1 \longrightarrow^\star e_3}\ \textsc{step}$$

Next, let's take a challenge problem from a previous set of notes. Figures 3 and 4 contain big-step and small-step semantics for arithmetic (Figure 2), respectively. Recall that a big-step semantics is ***adequate*** w.r.t. a small-step semantics if $e \Downarrow v$ implies $e \longrightarrow^\star v$.

Proposition 2. *(Adequacy) $e \Downarrow v$ implies $e \longrightarrow^\star v$ (as defined in Figures 3 and 4).*

Proof. Let's prove this by induction on the derivation $\mathfrak{D}$ of $e \Downarrow v$. There are four rules in the big-step semantics, so there are four possible ways the derivation of $e \Downarrow v$ ends.

Case 1. (INTLIT) Suppose $\mathfrak{D}$ ends with

$$\frac{e_1 \longrightarrow e_2}{e_1 \mathbin{+} e \longrightarrow e_2 \mathbin{+} e}\ \textsc{addRedLeft} \qquad \frac{e_1 \longrightarrow e_2}{e \mathbin{+} e_1 \longrightarrow e \mathbin{+} e_2}\ \textsc{addRedRight}$$

$$\frac{m \in \mathbb{Z} \qquad n \in \mathbb{Z}}{m \mathbin{+} n \longrightarrow m + n}\ \textsc{addLits} \qquad \frac{e_1 \longrightarrow e_2}{e_1 \mathbin{-} e \longrightarrow e_2 \mathbin{-} e}\ \textsc{subRedLeft}$$

$$\frac{e_1 \longrightarrow e_2}{e \mathbin{-} e_1 \longrightarrow e \mathbin{-} e_2}\ \textsc{subRedRight} \qquad \frac{m \in \mathbb{Z} \qquad n \in \mathbb{Z}}{m \mathbin{-} n \longrightarrow m - n}\ \textsc{subLits}$$

$$\frac{e_1 \longrightarrow e_2}{e_1 \mathbin{*} e \longrightarrow e_2 \mathbin{*} e}\ \textsc{mulRedLeft} \qquad \frac{e_1 \longrightarrow e_2}{e \mathbin{*} e_1 \longrightarrow e \mathbin{*} e_2}\ \textsc{mulRedRight}$$

$$\frac{m \in \mathbb{Z} \qquad n \in \mathbb{Z}}{m \mathbin{*} n \longrightarrow m \times n}\ \textsc{mulLits}$$

Figure 4: A small-step semantics for arithmetic expressions

$$\frac{}{n \Downarrow n}\ \textsc{intLit}$$

where $n \in \mathbb{Z}$. In particular, this is the *entirety* of the derivation. Then we need to show $n \longrightarrow^\star n$. This is easy:

$$\frac{}{n \longrightarrow^\star n}\ \textsc{refl}$$

Case 2. (ADD) Suppose $\mathfrak{D}$ ends with

$$\frac{\begin{array}{cc}[\mathfrak{D}_1] & [\mathfrak{D}_2] \\ e_1 \Downarrow v_1 & e_2 \Downarrow v_2\end{array}}{e_1 \mathbin{+} e_2 \Downarrow v_1 + v_2}\ \textsc{add}$$

By the induction hypothesis, we get that $e_1 \longrightarrow^\star v_1$ and $e_2 \longrightarrow^\star v_2$ (because $e_1 \Downarrow v_1$ and $e_2 \Downarrow v_2$ are derivable judgments). Then we have to find a way to combine the derivations of these judgments into a single derivation for $e_1 \mathbin{+} e_2 \longrightarrow^\star v_1 + v_2$. First, we need to show:

$$e_1 \longrightarrow^\star e_2 \qquad \text{implies} \qquad e_1 \mathbin{+} e \longrightarrow^\star e_2 \mathbin{+} e$$

and

$$e_1 \longrightarrow^\star e_2 \qquad \text{implies} \qquad e \mathbin{+} e_1 \longrightarrow^\star e \mathbin{+} e_2$$

We leave this as an exercise. We use the first claim above to show that $e_1 \mathbin{+} e_2 \longrightarrow^\star v_1 \mathbin{+} e_2$ and the second to then show $v_1 \mathbin{+} e_2 \longrightarrow^\star v_1 \mathbin{+} v_2$. Then by Proposition 1 above, we have

$$e_1 \mathbin{+} e_2 \longrightarrow^\star v_1 \mathbin{+} v_2$$

$$
\begin{array}{rcl}
e & ::= & \lambda x^{\tau}.e \mid ee \mid x \\
\tau & ::= & \bot \mid \tau \to \tau
\end{array}
$$

Figure 5: Abstract syntax for STLC

$$
\frac{(x : \tau) \in \Gamma}{\Gamma \vdash x : \tau}\ \textsc{var} \qquad \frac{\Gamma, x : \tau_1 \vdash e : \tau_2}{\Gamma \vdash \lambda x^{\tau_1}.e : \tau_1 \to \tau_2}\ \textsc{fun}
$$

$$
\frac{\Gamma \vdash e_1 : \tau_2 \to \tau \qquad \Gamma \vdash e_2 : \tau_2}{\Gamma \vdash e_1 e_2 : \tau}\ \textsc{app}
$$

Figure 6: Type system for STLC

by composing our two multi-step derivations. It's not difficult to show that

$$
e_1 \longrightarrow^{\star} e_2 \qquad \text{and} \qquad e_2 \longrightarrow e_3 \qquad \text{implies} \qquad e_1 \longrightarrow^{\star} e_3
$$

which we leave as another exercise. Since $v_1 + v_2 \longrightarrow v_1 + v_2$ by the ADDLITS rule, we finally have that $e_1 + e_2 \longrightarrow^{\star} v_1 + v_2$, as desired.

Case 3. (SUB) Nearly identical to Case 2. Left as an exercise.

Case 4. (MUL) Nearly identical to Case 2. Left as an exercise. □

Exercise 1. *Complete the proof above. For practice, I'd recommend writing the whole thing from scratch.*

Exercise 2. *Prove that $e_1 \longrightarrow^{\star} e_2$ implies $e_1 + e \longrightarrow^{\star} e_2 + e$ according to the rules in Figure 4. You should do this by induction on the derivation of $e_1 \longrightarrow^{\star} e_2$.*

Exercise 3. *Show that for any small-step semantics if $e_1 \longrightarrow^{\star} e_2$ and $e_2 \longrightarrow e_3$, then $e_1 \longrightarrow^{\star} e_3$.*

Progress and Preservation of STLC. The syntax, typing, and semantic rules for the simply-typed λ-calculus are given in Figures 5, 6, and 7, respectively.

$$
\frac{e_1 \longrightarrow e_2}{e_1 e \longrightarrow e_2 e}\ \textsc{appLeft} \qquad \frac{}{(\lambda x^{\tau}.e_1)e_2 \longrightarrow [e_2/x]e_1}\ \textsc{beta}
$$

$$
v \ ::= \ \lambda x^{\tau}.e
$$

Figure 7: Call-by-name small-step semantics for STLC

We'll be using CBN semantics here, but the choice doesn't matter terribly (it has fewer rules). Our goal in what remains is to show that STLC satisfies progress and preservation.

As a warm-up, we show the well-scopedness of well-typed expressions. An expression is ***well-scoped*** if all of its free variables are declared.

Theorem 1. *If $\Gamma \vdash e : \tau$ and $x \in \mathsf{FV}(e)$, then there is a type τ' such that $(x : \tau') \in \Gamma$.*

Proof. We prove this by induction on the typing derivation $\mathfrak{D}$ of $\Gamma \vdash e : \tau$. This means considering all the possible cases of the last rule in the given derivation.

Case 1. (VAR) Suppose $\mathfrak{D}$ ends with

$$\frac{}{\Gamma \vdash x : \tau}\ \textsc{var}$$

In particular, $\mathfrak{D}$ is a single axiom, and the side condition of the VAR rule ensures that $(x : \tau) \in \Gamma$, as desired (note that x is the only free variable of x, i.e., $\mathsf{FV}(x) = \{x\}$).

Case 2. (FUN)

Suppose $\mathfrak{D}$ ends with

$$\frac{\begin{array}{c}[\mathfrak{D}'] \\ \Gamma, x : \tau_1 \vdash e' : \tau_2\end{array}}{\Gamma \vdash \lambda x^{\tau_1}.e' : \tau_1 \to \tau_2}\ \textsc{fun}$$

Recall that $\mathsf{FV}(\lambda x^{\tau_1}.e') = \mathsf{FV}(e') \setminus \{x\}$. So $y \in \mathsf{FV}(\lambda x^{\tau_1}.e')$ implies $y \in \mathsf{FV}(e')$ and $y \neq x$. By the inductive hypothesis, we know that every free variable of e' is declared in $\Gamma, x : \tau_1$. Since $y \neq x$, it must be that y is declared in Γ (as desired).

Case 3. (APP) Suppose $\mathfrak{D}$ ends with

$$\frac{\begin{array}{cc}[\mathfrak{D}_1] & [\mathfrak{D}_2] \\ \Gamma \vdash e_1 : \tau_2 \to \tau & \Gamma \vdash e_2 : \tau_2\end{array}}{\Gamma \vdash e_1 e_2 : \tau}\ \textsc{app}$$

By definition, $\mathsf{FV}(e_1 e_2) = \mathsf{FV}(e_1) \cup \mathsf{FV}(e_2)$. In other words, if y is free in $e_1 e_2$, then it's free in e_1 or e_2. If y is free in e_1, then by the induction hypothesis applied to the left premise, y is declared in Γ. If y is free in e_2, then by the induction hypothesis applied to the right premise, y is declared in Γ. In either case, y is declared in Γ. □

To some degree, this shouldn't be surprising. The context exists to keep track of the free variables that appear in a given expression. But this isn't something that we get "for free;" we have to prove it.

Exercise 4. *Demonstrate that the converse does not hold, i.e., it's not the case that if $\Gamma \vdash e : \tau$ and x is declared in Γ then $x \in \mathsf{FV}(e)$.*

Onto the main event.

Theorem 2. *(Progress) If $\varnothing \vdash e : \tau$ for some type τ, then e is a value or there is an expression e' such that $e \longrightarrow e'$.*

Proof. We proceed by induction on the derivation $\mathfrak{D}$ of $\varnothing \vdash e : \tau$. There are three typing rules for STLC that could end $\mathfrak{D}$, but only two apply. The VAR rule can't end $\mathfrak{D}$ because the context is empty, and the side condition for the VAR rule can't be satisfied.

Case 1. (VAR) **Not possible.**

Case 2. (FUN) Suppose $\mathfrak{D}$ ends with

$$\frac{\begin{array}{c}[\mathfrak{D}'] \\ \{x : \tau_1\} \vdash e' : \tau_2\end{array}}{\varnothing \vdash \lambda x^{\tau_1} e' : \tau_1 \to \tau_2}\ \textsc{fun}$$

where $e = \lambda x^{\tau_1} e'$ and $\tau = \tau_1 \to \tau_2$. Then e is a value according to the semantics.

Case 3. (APP) Suppose $\mathfrak{D}$ ends with

$$\frac{\begin{array}{cc}[\mathfrak{D}_1] & [\mathfrak{D}_2] \\ \varnothing \vdash e_1 : \tau_2 \to \tau & \varnothing \vdash e_2 : \tau_2\end{array}}{\varnothing \vdash e_1 e_2 : \tau}\ \textsc{app}$$

By the induction hypothesis applied to $\mathfrak{D}_1$, we know that e_1 is either a value or there is an expression e_1' such that $e_1 \longrightarrow e_1'$.

- *Sub-case 3.1.* Suppose e_1 is a value of the form $\lambda x^{\tau'}.e'$. Then

  $$e_1 e_2 = (\lambda x^{\tau'}.e')e_2 \longrightarrow [e_2/x]e'$$

  by the BETA rule.

- *Sub-case 3.2.* Suppose there is an expression e_1' such that $e_1 \longrightarrow e_1'$. Then

  $$e_1 e_2 \longrightarrow e_1' e_2$$

  by the APPLEFT rule.

In either sub-case, we're able to make progress on $e_1 e_2$. □

Progress tends to be the easier of the two. It's essentially well-scopedness that ensures applications never get stuck (because the left expression of an application can never be a lone variable). We'll have to work a bit harder for preservation, but we'll first attempt the proof in the "expected" way to see why.

Theorem 3. *(Preservation) If $\Gamma \vdash e : \tau$ and $e \longrightarrow e'$ then $\Gamma \vdash e' : \tau$.*

Proof. As you might expect, we proceed by induction on the derivation $\mathfrak{D}$ of $\Gamma \vdash e : \tau$. There are three rules that could end this derivation.

Case 1. (VAR) Suppose the derivation ends with

$$\frac{}{\Gamma \vdash x : \tau}\ \textsc{var}$$

where $e = x$. Variables can't be reduced, so there's nothing to show.

Case 2. (FUN) Suppose the derivation ends with

$$\frac{\begin{array}{c}[\mathfrak{D}'] \\ \Gamma, x : \tau_1 \vdash e' : \tau_2\end{array}}{\Gamma \vdash \lambda x^{\tau_1}.e' : \tau_1 \to \tau_2}\ \textsc{fun}$$

where $e = \lambda x^{\tau_1}.e'$ and $\tau = \tau_1 \to \tau_2$. Functions can't be reduced, so there's nothing to show.

Case 3. (APP) Suppose the derivation ends with

$$\frac{\begin{array}{cc}[\mathfrak{D}_1] & [\mathfrak{D}_2] \\ \Gamma \vdash e_1 : \tau_2 \to \tau & \Gamma \vdash e_2 : \tau_2\end{array}}{\Gamma \vdash e_1 e_2 : \tau}\ \textsc{app}$$

where $e = e_1 e_2$. There are two ways $e_1 e_2$ can be reduced. Just as before, we proceed *again* by induction on derivations, this time over the derivation $\mathfrak{E}$ of $e_1 e_2 \longrightarrow e'$.

- *Sub-case 3.1.* (APPLEFT) Suppose $\mathfrak{E}$ ends with

$$\frac{\begin{array}{c}[\mathfrak{E}'] \\ e_1 \longrightarrow e_1'\end{array}}{e_1 e_2 \longrightarrow e_1' e_2}\ \textsc{appLeft}$$

  where $e' = e_1' e_2$. Since $\Gamma \vdash e_1 : \tau_2 \to \tau$ and $e_1 \longrightarrow e_1'$, by the induction hypothesis, there is a derivation $\mathfrak{D}_1'$ of $\Gamma \vdash e_1' : \tau_2 \to \tau$, i.e., the type of e_1 is preserved after reduction to e_1'. We can then combine $\mathfrak{D}_1'$ and $\mathfrak{D}_2$ using the APP rule.

$$\frac{\begin{array}{cc}[\mathfrak{D}_1'] & [\mathfrak{D}_2] \\ \Gamma \vdash e_1' : \tau_2 \to \tau & \Gamma \vdash e_2 : \tau_2\end{array}}{\Gamma \vdash e_1' e_2 : \tau}\ \textsc{app}$$

  So $e' = e_1' e_2$ has the same type as $e = e_1 e_2$ under Γ.

- *Sub-case 3.2.* (BETA) Suppose the derivation of $e_1 e_2 \longrightarrow e'$ ends in

$$\frac{}{(\lambda x^{\tau'} e_1') e_2 \longrightarrow [e_2/x]e_1'}\ \textsc{beta}$$

  where $e_1 = \lambda x^{\tau'} e_1'$ and $e' = [e_2/x]e_1'$. Our goal then is to show that $\Gamma \vdash [e_2/x]e_1' : \tau$, but this seems difficult; there isn't anything around that lets us just recombine things as we've done before...

Substitution is troublesome. On the one hand, the impact of the β-rule is localized to the redex that's reduced, but the redex itself changes fairly drastically; variables in the AST are replaced with whole new sub-ASTs. But, when in doubt, turn your challenge into your goal. Here's a lemma, often called, naturally enough, the **substitution lemma**.

Lemma 1. *If* $\Gamma, x : \tau' \vdash e : \tau$ *and* $\Gamma \vdash e' : \tau'$*, then* $\Gamma \vdash [e'/x]e : \tau$*.*

Proof. We show this by, you guessed it, induction on derivations. Let $\mathfrak{D}$ denote the derivation of $\Gamma, x : \tau' \vdash e : \tau$ and $\mathfrak{D}'$ the derivation of $\Gamma \vdash e' : \tau'$. We proceed by induction on $\mathfrak{D}$.

Case 1. (VAR) Suppose $\mathfrak{D}$ ends with:

$$\frac{}{\Gamma, x : \tau' \vdash y : \tau}\ \textsc{var}$$

where y is a variable and $e = y$. In particular, this is the entirety of $\mathfrak{D}$. By the side condition of the VAR rule, we know that $(y : \tau) \in \Gamma, x : \tau'$. There are two sub-cases to consider.

- *Sub-case 1.1.* Suppose $y = x$. Since x is declared to be of type τ' in $\Gamma, x : \tau'$, it must also be that $\tau = \tau'$. Then $[e'/x]y = [e'/x]x = e'$, and our goal becomes to show that $\Gamma \vdash e' : \tau'$, which is derived by $\mathfrak{D}'$.
- *Sub-case 1.2.* Suppose $y \neq x$. Then it must be that $(y : \tau) \in \Gamma$ and $[e'/x]y = y$. But then we can derive $\Gamma \vdash y : \tau$ by the VAR rule.

$$\frac{}{\Gamma \vdash y : \tau}\ \textsc{var}$$

Case 2. (FUN) Suppose $\mathfrak{D}$ ends with

$$\frac{\begin{array}{c}[\mathfrak{D}_1] \\ \Gamma, x : \tau', y : \tau_1 \vdash e_1 : \tau_2\end{array}}{\Gamma, x : \tau' \vdash \lambda y^{\tau_1}.e_1 : \tau_1 \to \tau_2}\ \textsc{fun}$$

where $e = \lambda y^{\tau_1}.e_1$ and $\tau = \tau_1 \to \tau_2$. There are two sub-cases to consider.

- *Sub-case 2.1.* Suppose $y = x$. Then

  $$\begin{aligned}\Gamma, x : \tau', y : \tau_1 &= \Gamma, x : \tau_1 \\ [e'/x]e &= [e'/x](\lambda x^{\tau_1}.e_1) = \lambda x^{\tau_1}.e_1\end{aligned}$$

  which means we can derive $\Gamma \vdash [e'/x]e : \tau$ by applying the FUN rule (with no appeal to the induction hypothesis).[2]

[2] Here we're being a bit hand-wavy. Because we take contexts to be sets, the "most recently added" x declaration takes precedence.

$$\frac{\begin{array}{c}[\mathfrak{D}_1]\\ \Gamma, x : \tau_1 \vdash e_1 : \tau_2\end{array}}{\Gamma \vdash \lambda x^{\tau_1}.e_1 : \tau_1 \to \tau_2}\ \textsc{fun}$$

- *Sub-case 2.2.* Suppose $y \neq x$. Then, $[e'/x]e = [e'/x](\lambda y^{\tau_1}.e_1) = \lambda y^{\tau_1}.[e'/x]e_1$. Since we take contexts to be *sets* of variable declarations, there is no difference between

  $$\Gamma, x : \tau', y : \tau_1 \vdash e_1 : \tau_2 \qquad \text{and} \qquad \Gamma, y : \tau_1, x : \tau' \vdash e_1 : \tau_2$$

  That is, we can freely move around declarations in our contexts. By the induction hypothesis, there is a derivation $\mathfrak{D}_1'$ of $\Gamma, y : \tau_1 \vdash [e'/x]e_1 : \tau_2$, which we can use with the FUN rule to derive $\Gamma \vdash [e'/x]e : \tau$.

  $$\frac{\begin{array}{c}[\mathfrak{D}_1']\\ \Gamma, y : \tau_1 \vdash [e'/x]e_1 : \tau_2\end{array}}{\Gamma \vdash \lambda y^{\tau_1}.[e'/x]e_1 : \tau_1 \to \tau_2}\ \textsc{fun}$$

Case 3. (APP) Suppose that $\mathfrak{D}$ ends with

$$\frac{\begin{array}{cc}[\mathfrak{D}_1] & [\mathfrak{D}_2]\\ \Gamma, x : \tau' \vdash e_1 : \tau_2 \to \tau & \Gamma, x : \tau' \vdash e_2 : \tau_2\end{array}}{\Gamma, x : \tau' \vdash e_1 e_2 : \tau}\ \textsc{app}$$

By the induction hypothesis applied to both premises, there is a derivation $\mathfrak{D}_1'$ of $\Gamma \vdash [e'/x]e_1 : \tau_2 \to \tau$ and a derivation $\mathfrak{D}_2'$ of $\Gamma \vdash [e'/x]e_2 : \tau_2$. Since $[e'/x]e = [e'/x](e_1 e_2) = ([e'/x]e_1)([e'/x]e_2)$, we can derive $\Gamma \vdash [e'/x]e : \tau$ using these two new derivations and the APP rule:

$$\frac{\begin{array}{cc}[\mathfrak{D}_1'] & [\mathfrak{D}_2']\\ \Gamma \vdash [e'/x]e_1 : \tau_2 \to \tau & \Gamma \vdash [e'/x]e_2 : \tau_2\end{array}}{\Gamma \vdash [e'/x](e_1 e_2) : \tau}\ \textsc{app}$$

□

Let's finish up our proof of preservation in STLC.

Proof. (of preservation in STLC) It's not difficult to show that if an anonymous function is well-typed in a given context, the only possible last rule in its derivation is FUN. We can use this fact to rewrite our derivation of $\Gamma \vdash e_1 e_2 : \tau$ in this case.

$$\frac{\begin{array}{cc}\dfrac{\begin{array}{c}[\mathfrak{D}_1'']\\ \Gamma, x : \tau_2 \vdash e_1' : \tau\end{array}}{\Gamma \vdash \lambda x^{\tau_2}.e_1' : \tau_2 \to \tau}\ \textsc{fun} & \begin{array}{c}[\mathfrak{D}_2]\\ \Gamma \vdash e_2 : \tau_2\end{array}\end{array}}{\Gamma \vdash (\lambda x^{\tau_2}.e_1')e_2 : \tau}\ \textsc{app}$$

In particular, we know that $e_1 = \lambda x^{\tau_2}.e_1'$ (with the new annotation) because that's the only way the FUN rule can be applied. But now we have $\Gamma, x : \tau_2 \vdash e_1' : \tau$ and $\Gamma \vdash e_2 : \tau_2$, so by the substitution lemma, we have $\Gamma \vdash [e_2/x]e_1' : \tau$.

That completes all cases. *Fin.* □

Exercise 5. *Show that if* $\Gamma \vdash \lambda x^{\tau'} e : \tau$, *then there is a type* τ'' *such that* $\tau = \tau' \to \tau''$ *and* $\Gamma, x : \tau' \vdash e : \tau''$. *This is sometimes called a* ***generation lemma****.*

Exercise 6. *Suppose we made a mistake when writing the* FUN *rule, and forgot to require the type annotation to match the argument type.*

$$\frac{\Gamma, x : \tau \vdash e : \tau_2}{\Gamma \vdash \lambda x^{\tau}.e : \tau_1 \to \tau_2}\ \textsc{funWrong}$$

Show that STLC with FUN *replaced by* FUNWRONG *does not satisfy preservation.*

Exercise 7. *Show that STLC with* FUN *replaced by* FUNWRONG *(as in the previous problem) satisfies progress.*

Exercise 8. *Suppose we consider a variation of STLC that requires applications to have two arguments.*

$$\begin{array}{rcl} e & ::= & \lambda x^{\tau}.e \mid e(e,e) \mid x \\ \tau & ::= & \bot \mid \tau \to \tau \\ v & ::= & \bullet \mid \lambda x^{\tau}.e \end{array}$$

$$\frac{(x : \tau) \in \Gamma}{\Gamma \vdash x : \tau}\ \textsc{var} \qquad \frac{\Gamma, x : \tau_1 \vdash e : \tau_2}{\Gamma \vdash \lambda x^{\tau_1}.e : \tau_1 \to \tau_2}\ \textsc{fun}$$

$$\frac{\Gamma \vdash e : \tau_1 \to \tau_2 \to \tau \qquad \Gamma \vdash e_1 : \tau_1 \qquad \Gamma \vdash e_2 : \tau_2}{\Gamma \vdash e(e_1, e_2) : \tau}\ \textsc{app}$$

$$\frac{e \longrightarrow e'}{e(e_1, e_2) \longrightarrow e'(e_1, e_2)}\ \textsc{appLeft}$$

$$\frac{}{(\lambda x^{\tau_1} \lambda y^{\tau_2}.e)(e_1, e_2) \longrightarrow [e_2/y][e_1/x]e}\ \textsc{beta}$$

Show that this variant of STLC does not satisfy progress.

Exercise 9. *Show that the variant of STLC from the previous problem satisfies preservation.*

Exercise 10. *Show that STLC satisfies progress and preservation using CBV semantics.*

Exercise 11. *Introduce integers, integer addition, Booleans, and if-expressions into STLC, and show that the resulting system satisfies progress and preservation.*

Exercise 12. *Show that if* $\varnothing \vdash e : \tau$ *and* $e \Downarrow v$ *via the big-step CBN semantics for STLC, then* $\varnothing \vdash v : \tau$.