1.6.3 How to attempt a metatheoretic proof

A metatheoretic proof works by induction on the structure of valid derivations. A derivation is a tree, and we can assume an induction hypothesis holds for any proper subtree—or, if you prefer, we can prove metatheorems by induction on the height of a derivation tree.

A valid derivation can end in any rule, so a proof by induction on a derivation’s structure has a case for every rule. In a language as big as Impcore, that’s a lot of cases. To make my proofs readable and convincing, I organize each case in exactly the same way. Here’s a template for a proof by structural induction over a derivation $D$:

1. When the last rule used in $D$ is $\text{RuleName}$, and $\text{RuleName}$ has conclusion $C$ and premises $P_1$ to $P_n$, the derivation must have the following form:

   \[
   D = \frac{P_1 \quad \cdots \quad P_n}{C} \text{ RuleName}
   \]

   Commentary: The conclusion $C$ is the evaluation judgment of which $D$ is a proof. If any particular $P_i$ is also an evaluation judgment, I write its derivation above it, as in

   \[
   D = \frac{D_1 \quad P_1 \quad \cdots \quad D_i \quad D_i \quad \cdots \quad D_n \quad P_n}{C} \text{ RuleName}
   \]

   A premise like “$x \notin \text{dom } \rho$” is not an evaluation judgment and is not supported by a subderivation.

2. The form of $e$ is syntactic form, and whatever additional analysis goes with that syntactic form and with rule $\text{RuleName}$.

3. Our obligation is to prove that the induction hypothesis holds for the judgment below the line. We must therefore prove whatever it is.

4. Identifying each premise $P_i$ that is an evaluation judgment, because derivation $D_i$ is strictly smaller than derivation $D$, we are permitted to assume that the induction hypothesis applies to derivation $D_i$. This assumption gives us whatever it gives us.

5. From the truth of premises $P_1$ to $P_n$, plus the information from the induction hypothesis, we show that the induction hypothesis holds for the judgment below the line.

6. Our obligation is met.

This template has served me well, but part of it may surprise you: it doesn’t distinguish between “base cases” and “inductive cases.” The distinction is there—a base case has no evaluation judgments above the line—but in a programming-language proof, the distinction is not terribly useful. For example, the base cases might or might not be easy, and they might or might not be the ones that fail.

The template is instantiated for every case in a proof. As a demonstration, I instantiate the template to try to prove the metatheoretic conjecture,

\[
\text{If an expression } e \text{ is evaluated successfully, then every variable in } e \text{ is defined.}
\]

(This conjecture isn’t actually true, but that’s a good thing—we learn the most from the things we try to prove that aren’t so.)
1.9. EXERCISES

\[
\langle \text{LITERAL}(v), \xi, \phi, \rho \rangle \Downarrow \langle v, \xi, \phi, \rho \rangle \quad \text{(LITERAL)}
\]

\[
x \in \text{dom } \rho
\]

\[
\langle \text{VAR}(x), \xi, \phi, \rho \rangle \Downarrow \langle \rho(x), \xi, \phi, \rho \rangle
\]

\[
x \notin \text{dom } \rho
\]

\[
\langle \text{VAR}(x), \xi, \phi, \rho \rangle \Downarrow \langle (\xi(x), \xi, \phi, \rho) \rangle
\]

\[
x \in \text{dom } \rho
\]

\[
\langle e, \xi, \phi, \rho \rangle \Downarrow \langle v, \xi, \phi, \rho' \rangle \quad \text{(FORMALASSIGN)}
\]

\[
x \notin \text{dom } \rho
\]

\[
\langle e, \xi, \phi, \rho \rangle \Downarrow \langle v, v', \xi, \phi, \rho' \rangle
\]

\[
\langle \text{set}(x, e), \xi, \phi, \rho \rangle \Downarrow \langle v, v', \xi, \phi, \rho' \rangle \quad \text{(GLOBALASSIGN)}
\]

\[
\langle e_1, \xi, \phi, \rho \rangle \Downarrow \langle v_1, \xi', \phi, \rho' \rangle \quad v_1 \neq 0
\]

\[
\langle e_2, \xi', \phi, \rho' \rangle \Downarrow \langle v_2, \xi', \phi, \rho'' \rangle
\]

\[
\langle e_3, \xi, \phi, \rho \rangle \Downarrow \langle v_3, \xi'', \phi, \rho'' \rangle
\]

\[
\langle \text{IF}(e_1, e_2, e_3), \xi, \phi, \rho \rangle \Downarrow \langle v_2, \xi', \phi, \rho'' \rangle
\]

\[
\langle e_1, \xi, \phi, \rho \rangle \Downarrow \langle v_1, \xi, \phi, \rho' \rangle \quad v_1 = 0
\]

\[
\langle e_2, \xi', \phi, \rho' \rangle \Downarrow \langle v_2, \xi', \phi, \rho'' \rangle
\]

\[
\langle e_3, \xi, \phi, \rho \rangle \Downarrow \langle v_3, \xi'', \phi, \rho'' \rangle
\]

\[
\langle \text{IF}(e_1, e_2, e_3), \xi, \phi, \rho \rangle \Downarrow \langle v_3, \xi'', \phi, \rho'' \rangle
\]

\[
\langle \text{WHILE}(e_1, e_2), \xi, \phi, \rho \rangle \Downarrow \langle v_3, \xi'', \phi, \rho'' \rangle
\]

\[
\langle e_1, \xi, \phi, \rho \rangle \Downarrow \langle v_1, \xi', \phi, \rho' \rangle \quad v_1 = 0
\]

\[
\langle \text{WHILE}(e_1, e_2), \xi, \phi, \rho \rangle \Downarrow \langle 0, \xi', \phi, \rho' \rangle
\]

\[
\langle \text{BEGIN}(), \xi, \phi, \rho \rangle \Downarrow \langle 0, \xi, \phi, \rho \rangle \quad \text{(EMPTYBEGIN)}
\]

\[
\langle e_1, \xi_0, \phi, \rho_0 \rangle \Downarrow \langle v_1, \xi_1, \phi, \rho_1 \rangle
\]

\[
\langle e_2, \xi_1, \phi, \rho_1 \rangle \Downarrow \langle v_2, \xi_2, \phi, \rho_2 \rangle
\]

\[
\vdots
\]

\[
\langle e_n, \xi_n-1, \phi, \rho_{n-1} \rangle \Downarrow \langle v_n, \xi_n, \phi, \rho_n \rangle
\]

\[
\langle \text{BEGIN}(e_1, e_2, \ldots, e_n), \xi_0, \phi, \rho_0 \rangle \Downarrow \langle v_n, \xi_n, \phi, \rho_n \rangle
\]

\[
\phi(f) = \text{USER}(\{x_1, \ldots, x_n\}, e)
\]

\[
x_1, \ldots, x_n \text{ all distinct}
\]

\[
\langle e_1, \xi_0, \phi, \rho_0 \rangle \Downarrow \langle v_1, \xi_1, \phi, \rho_1 \rangle
\]

\[
\vdots
\]

\[
\langle e_n, \xi_n-1, \phi, \rho_{n-1} \rangle \Downarrow \langle v_n, \xi_n, \phi, \rho_n \rangle
\]

\[
\langle e, \xi_n, \phi, \{x_1 \mapsto v_1, \ldots, x_n \mapsto v_n\} \rangle \Downarrow \langle v, \xi', \phi, \rho' \rangle
\]

\[
\langle \text{APPLY}(f, e_1, \ldots, e_n), \xi_0, \phi, \rho_0 \rangle \Downarrow \langle v, \xi', \phi, \rho' \rangle
\]  

Figure 1.5: Summary of operational semantics (expressions)