Topic 1. Substitution and scoping redux: Let expressions in D. We can define Let declaration forms in D via syntactic sugaring (how would you do it?), but it is convenient to include it as a primitive expression form. Like function application, we base the semantics of Let on substitution. Since this is a simpler expression form, it is perhaps easier to understand how substitution works by examining this rule form.

The syntax and scoping rule of Let expressions is specified as follows:

\[ e ::= \cdots \mid \text{Let } x = e \text{ In } e \] expressions (extended)

Definition 1.1 (Scope (extended)) The scope of \( x \) in \( \text{Let } x = e_1 \text{ In } e_2 \) is \( e_2 \).

Substitution is thus implicitly extended as follows:

\[
\begin{align*}
(\text{Let } x = e_1 \text{ In } e_2)[v/x] &= \text{Let } x = e_1[v/x] \text{ In } e_2 \\
(\text{Let } y = e_1 \text{ In } e_2)[v/x] &= \text{Let } y = e_1[v/x] \text{ In } e_2[v/x] \quad x \neq y
\end{align*}
\]

The semantics of Let expressions is then given as follows:

\[
\begin{align*}
\text{LET } e_1 \Rightarrow v' & \quad e_2[v'/x] \Rightarrow v \\
\text{Let } x = e_1 \text{ In } e_2 \Rightarrow v
\end{align*}
\]

Example:

\[
\begin{array}{c}
10 \Rightarrow 10 \\
2 + 10 \Rightarrow 12 \\
\hline
\text{Let } x = 10 \text{ In } 2 + x \Rightarrow 12
\end{array}
\]

since \((2 + x)[10/x] \equiv 2 + 10\).

Topic 2. Properties of D. Despite its simplicity, it turns out that D is Turing complete. Thus, it is not surprising that normalization fails for D. Note that the interesting counterexample, illustrating non-termination, employs self-reference, which also drives recursive programming.

Proposition 1.1 D is deterministic; if \( e \Rightarrow v \) and \( e \Rightarrow v' \) then \( v = v' \).

Proof. By structural induction on the derivation of \( e \Rightarrow v \). Note that for every syntactic form, there is only one rule that applies.

Proposition 1.2 D is not normalizing; i.e., it is not the case that for all \( e \) there exist \( v \) such that \( e \Rightarrow v \).

Proof. It suffices to provide an example. (True 1) is an example, since this expression is stuck—that is, is type mismatched—with no applicable evaluation rules. Another example is the expression:

\[ \Omega \triangleq (\text{Function } x \rightarrow xx)(\text{Function } x \rightarrow xx) \]

This expression is not normalizing, since any attempt to evaluate the expression results in the expression itself:

\[ (xx)[(\text{Function } x \rightarrow xx)/x] = (\text{Function } x \rightarrow xx)(\text{Function } x \rightarrow xx) \]

Therefore, this expression can never evaluate to a value.