1 Soundness from the Operational Perspective

We have seen that we can write useful typing rules, but how do we know we got them right? This depends on what the type system is meant to accomplish. The traditional application is avoiding runtime type errors. We say that a type system is *sound* if well-typed programs do not incur runtime type errors; that is, they do not get stuck when evaluated according to the operational semantics:

$$\textit{The typing rules are sound} \overset{\triangle}{\iff} \textit{no well-typed program gets stuck.}$$

To be more precise, let us call e *irreducible* and write *Irred*(e) if there is no reduction possible on e. All values of $\lambda^{\to}$ are irreducible. If e is irreducible but is not a value, then e is said to be *stuck*. We wish to show

Theorem 1 (Operational Soundness). *If $\vdash e : \tau$ and $e \xrightarrow{*} e'$ and $\mathit{Irred}(e')$, then $e' \in \mathit{Val}$ and $\vdash e' : \tau$.*

We will prove this in two steps using the following two lemmas:

Lemma 2 (Type Preservation). *If $\Gamma \vdash e : \tau$ and $e \to e'$, then $\Gamma \vdash e' : \tau$.*

Lemma 3 (Progress). *If $\vdash e : \tau$ and $\mathit{Irred}(e)$, then $e \in \mathit{Val}$.*

The type preservation lemma (Lemma 2) says that as we evaluate a program, its type is preserved at each step. The progress lemma (Lemma 3) says that every program is either a value or can be stepped to another program (and by type preservation, this will be of the same type).

Operational soundness follows easily from these two lemmas. Type preservation says every step preserves the type, so we use induction on the number of steps taken in $e \xrightarrow{*} e'$ to show that e' must have the same type as e. Then progress can be applied to e' to show that the evaluation is not stuck there. We will now set out to prove these two lemmas.

2 Proof of the Type Preservation Lemma

Assuming that $\Gamma \vdash e : \tau$ and $e \to e'$, we wish to show that $\Gamma \vdash e' : \tau$. We will do this by induction on the small-step operational semantics rules.

If $e \to e'$, there are three cases corresponding to the three evaluation rules:

$$\frac{e_0 \to e_0'}{e_0\, e_1 \to e_0'\, e_1}\ (\mathrm{L}) \qquad \frac{e \to e'}{v\, e \to v\, e'}\ (\mathrm{R}) \qquad \frac{}{(\lambda x : \sigma .\, e)\, v \to e\{v/x\}}\ (\beta)$$

- Case (L): $e_0\, e_1 \to e_0'\, e_1$.

  Because we have a typing derivation for $e_0\, e_1$, we know that there are typing derivations for e_0 and e_1 too. We must have $\Gamma \vdash e_0 : \sigma \to \tau$ and $\Gamma \vdash e_1 : \sigma$ for some type σ. By the induction hypothesis, the reduction $e_0 \to e_0'$ also preserves type, so $\Gamma \vdash e_0' : \sigma \to \tau$. Applying the typing rule for applications, we have that $\Gamma \vdash e_0'\, e_1 : \tau$.

- Case (R): $v\, e \to v\, e'$.

  This case is symmetrical to case (L). In this case it is the right-hand subexpression making the step.

- Case (β): $(\lambda x : \sigma .\, e)\, v \to e\{v/x\}$.

  The typing derivation of $\Gamma \vdash (\lambda x : \sigma .\, e)\, v : \tau$ must look like this:

$$\dfrac{\dfrac{\Gamma,\, x : \sigma \vdash e : \tau}{\Gamma \vdash (\lambda x : \sigma .\, e) : \sigma \to \tau} \quad \Gamma \vdash v : \sigma}{\Gamma \vdash (\lambda x : \sigma .\, e)\, v : \tau}$$

  We want to show that $\Gamma \vdash e\{v/x\} : \tau$ using the facts $\Gamma,\, x : \sigma \vdash e : \tau$ and $\vdash v : \sigma$. Our induction hypothesis does not help us here; we need to prove this separately. It follows as a special case of the substitution lemma below, which captures the type preservation of β-reduction.

3 The Substitution Lemma

Lemma 4 (Substitution Lemma)**.** $\vdash v : \sigma \;\Rightarrow\; (\Gamma,\, x : \sigma \vdash e : \tau \;\Leftrightarrow\; \Gamma \vdash e\{v/x\} : \tau)$.

We will prove this by structural induction on e.

Case 1 $x \notin FV(e)$.

This case covers the base cases $e \in \{n, \mathsf{true}, \mathsf{false}, \mathsf{null}\}$ and $e = y \neq x$ and λ-abstractions $\lambda x : \rho .\, e$ that bind x. In this case the substitution has no effect and any binding of x in the type environment Γ is irrelevant, thus the lemma reduces to the trivial statement

$$\vdash v : \sigma \;\Rightarrow\; (\Gamma \vdash e : \tau \;\Leftrightarrow\; \Gamma \vdash e : \tau).$$

Case 2 $e = x$.

In this case the lemma reduces to

$$\vdash v : \sigma \;\Rightarrow\; (\Gamma,\, x : \sigma \vdash x : \tau \;\Leftrightarrow\; \Gamma \vdash v : \tau),$$

since $x\{v/x\} = v$. Since v is closed, the type environment Γ is irrelevant, so the statement further reduces to

$$\vdash v : \sigma \;\Rightarrow\; (x : \sigma \vdash x : \tau \;\Leftrightarrow\; \vdash v : \tau).$$

Since types are unique, both sides of the double implication say that $\sigma = \tau$, so again the lemma reduces to a tautology.

Case 3 $e = e_0\, e_1$.

Suppose $\vdash v : \sigma$.

$$\begin{array}{rcll}
\Gamma,\, x : \sigma \vdash e_0\, e_1 : \tau & \Leftrightarrow & \exists \rho\; \Gamma,\, x : \sigma \vdash e_0 : \rho \to \tau \;\wedge\; \Gamma,\, x : \sigma \vdash e_1 : \rho & \text{typing rule for applications} \\
& \Leftrightarrow & \exists \rho\; \Gamma \vdash e_0\{v/x\} : \rho \to \tau \;\wedge\; \Gamma \vdash e_1\{v/x\} : \rho & \text{induction hypothesis} \\
& \Leftrightarrow & \Gamma \vdash (e_0\{v/x\})\,(e_1\{v/x\}) : \tau & \text{typing rule for applications} \\
& \Leftrightarrow & \Gamma \vdash (e_0\, e_1)\{v/x\} : \tau & \text{definition of substitution.}
\end{array}$$

Case 4 $e = \lambda y : \rho.\, e'$, where $y \neq x$ (the case $y = x$ was covered in Case 1).

Suppose $\vdash v : \sigma$. The type of $\lambda y : \rho.\, e'$, if it exists, must be $\rho \to \tau$ for some τ. Similarly, the type of $(\lambda y : \rho.\, e')\{v/x\} = \lambda y : \rho.\, (e'\{v/x\})$, if it exists, must be $\rho \to \tau'$ for some τ'.

$$\begin{array}{rcll}
\Gamma,\, x : \sigma \vdash (\lambda y : \rho.\, e') : \rho \to \tau & \Leftrightarrow & \Gamma,\, x : \sigma,\, y : \rho \vdash e' : \tau & \text{typing rule for abstractions} \\
& \Leftrightarrow & \Gamma,\, y : \rho,\, x : \sigma \vdash e' : \tau & \text{exchange} \\
& \Leftrightarrow & \Gamma,\, y : \rho \vdash e'\{v/x\} : \tau & \text{induction hypothesis} \\
& \Leftrightarrow & \Gamma \vdash \lambda y : \rho.\, (e'\{v/x\}) : \rho \to \tau & \text{typing rule for abstractions} \\
& \Leftrightarrow & \Gamma \vdash (\lambda y : \rho.\, e')\{v/x\} : \rho \to \tau & \text{definition of substitution.}
\end{array}$$

4 Proof of the Progress Lemma

To finish the proof of soundness, it remains to prove the progress lemma. Recall that this lemma states

$$\vdash e : \tau \;\wedge\; \mathit{Irred}(e) \;\Rightarrow\; e \in \mathit{Val},$$

or equivalently,

$$\vdash e : \tau \;\wedge\; e \notin \mathit{Val} \;\Rightarrow\; \exists e'\; e \to e'.$$

In other words, we cannot get stuck when evaluating a well-typed expression.

We prove the progress lemma using structural induction on e. Recall the definition of a term in $\lambda^{\to}$:

$$e \;::=\; b \;\mid\; x \;\mid\; \lambda x : \tau.\, e \;\mid\; e_0\, e_1,$$

where b denotes a constant. This gives four cases:

Case 1 $e = b$.

Since $b \in \mathit{Val}$, we are done.

Case 2 $e = x$.

This case is impossible, because we cannot assign a type to x if the type environment is empty.

Case 3 $e = \lambda x : \sigma.\, e'$.

This case requires another lemma:

Lemma 5. *If* $\Gamma \vdash e : \tau$ *then* $FV(e) \subseteq \mathsf{dom}\,\Gamma$.

We leave the proof as an exercise. Since $\vdash e : \tau$, it follows that e is closed, therefore is a value.

Case 4 $e = e_0\, e_1$.

We cannot have a value of this form, so the statement of the lemma reduces to

$$\vdash (e_0\, e_1) : \tau \;\Rightarrow\; \exists e'\; (e_0\, e_1) \to e'.$$

In any type derivation of $\vdash (e_0\,e_1) : \tau$, the last step must have the form

$$\frac{\vdash e_0 : \sigma \to \tau \quad \vdash e_1 : \sigma}{\vdash (e_0\,e_1) : \tau}$$

for some type σ. By the induction hypothesis, either $e_0 \in \mathit{Val}$ or $\exists e_0'\ e_0 \to e_0'$, and either $e_1 \in \mathit{Val}$ or $\exists e_1'\ e_1 \to e_1'$. This gives three possibilities:

- If e_0 is not a value, then by the induction hypothesis there $\exists e_0'\ e_0 \to e_0'$, therefore

  $$\frac{e_0 \to e_0'}{e_0\,e_1 \to e_0'\,e_1},$$

  so $e = e_0\,e_1$ can be further reduced.

- If e_0 is a value v but e_1 is not a value, then by the induction hypothesis $\exists e_1'\ e_1 \to e_1'$, and we have

  $$\frac{e_1 \to e_1'}{v\,e_1 \to v\,e_1'},$$

  so $e = v\,e_1$ can be further reduced.

- If both e_0 and e_1 are values, then since e_0 is a value with an arrow type $\sigma \to \tau$, it has to be an abstraction, say $e_0 = \lambda x : \sigma .\, e'$, and e_1 is some value v of type σ. Then

  $$e \;=\; (\lambda x : \sigma .\, e')\,v \to e'\,\{v/x\},$$

  so e can be further reduced.

This completes the proof.