Evaluation Contexts

Stefan Muller*

CSE 5095: Types and Programming Languages, Fall 2025
Lecture 22

Unlike the other day when we saw big-step semantics, today we're going to see a way of presenting semantics that's entirely equivalent to small-step semantics and so has the same pros and cons, but lets us specify "only" the "interesting" rules (scare quotes are there because, as is often the case, we don't actually get rid of the complexity, we just move it somewhere else). We'll consider SLTC again, plus "evaluation contexts" which represent expressions with exactly one "hole", written $\circ$ (or sometimes []).

Types	τ	::=	$\mathsf{unit} \mid \tau \to \tau \mid \tau \times \tau \mid \tau + \tau$
Values	v	::=	$() \mid \lambda x : \tau.e \mid (v, v) \mid \mathsf{inl}\ v \mid \mathsf{inr}\ v$
Expressions	e	::=	$v \mid x \mid e\ e \mid (e, e) \mid \mathsf{fst}\ e \mid \mathsf{snd}\ e \mid \mathsf{inl}\ e \mid \mathsf{inr}\ e \mid \mathsf{case}\ e\ \mathsf{of}\ \{x.e; y.e\}$
Evaluation Contexts	$\mathcal{E}$	::=	$\circ \mid \mathcal{E}\ e \mid v\ \mathcal{E} \mid (\mathcal{E}, e) \mid (v, \mathcal{E}) \mid \mathsf{fst}\ \mathcal{E} \mid \mathsf{snd}\ \mathcal{E} \mid \mathsf{inl}\ \mathcal{E} \mid \mathsf{inr}\ \mathcal{E} \mid \mathsf{case}\ \mathcal{E}\ \mathsf{of}\ \{x.e; y.e\}$

Note that the hole in an evaluation context is always in the part of the expression that's allowed to step (using search rules). This lets us separate an expression into "the part that can step" and "everything else." We write $\mathcal{E}[e]$ to mean filling the hole in $\mathcal{E}$ with e:

$$
\begin{array}{lcl}
\circ[e] & = & e \\
(\mathcal{E}\ e)[e'] & = & (\mathcal{E}[e'])\ e \\
(v\ \mathcal{E})[e'] & = & v\ (\mathcal{E}[e']) \\
(\mathcal{E}, e)[e'] & = & (\mathcal{E}[e'], e) \\
(v, \mathcal{E})[e'] & = & (v, \mathcal{E}[e']) \\
(\mathsf{fst}\ \mathcal{E})[e] & = & \mathsf{fst}\ (\mathcal{E}[e]) \\
(\mathsf{snd}\ \mathcal{E})[e] & = & \mathsf{snd}\ (\mathcal{E}[e]) \\
(\mathsf{inl}\ \mathcal{E})[e] & = & \mathsf{inl}\ (\mathcal{E}[e]) \\
(\mathsf{inr}\ \mathcal{E})[e] & = & \mathsf{inr}\ (\mathcal{E}[e]) \\
(\mathsf{case}\ \mathcal{E}\ \mathsf{of}\ \{x.e_1; y.e_2\})[e] & = & \mathsf{case}\ \mathcal{E}[e]\ \mathsf{of}\ \{x.e_1; y.e_2\}
\end{array}
$$

For example,

$$(5, (6, (\lambda x : \mathsf{int}.x)\ (\mathsf{fst}\ (7, 8)))) = (5, (6, (\lambda x : \mathsf{int}.x)\ \circ))[\mathsf{fst}\ (7, 8)]$$

Our step judgment now just has one rule that lets you step the expression that's filling the hole!

$$\frac{e \to e'}{\mathcal{E}[e] \mapsto \mathcal{E}[e']}\ (\textsc{StepExp})$$

We still need to define the judgment $e \to e'$, but it only has the interesting rules from our step judgment. All of the search rules are handled by the fact that the evaluation context represents searching through the expression to find the part that can step.

$$\frac{}{(\lambda x : \tau.e)\ v \to [v/x]e}\ (\textsc{StepApp}) \qquad \frac{}{\mathsf{fst}\ (v_1, v_2) \to v_1}\ (\textsc{StepFst}) \qquad \frac{}{\mathsf{snd}\ (v_1, v_2) \to v_2}\ (\textsc{StepSnd})$$

$$\frac{}{\mathsf{case}\ \mathsf{inl}\ v\ \mathsf{of}\ \{x.e_1; y.e_2\} \to [v/x]e_1}\ (\textsc{StepCaseL}) \qquad \frac{}{\mathsf{case}\ \mathsf{inr}\ v\ \mathsf{of}\ \{x.e_1; y.e_2\} \to [v/y]e_2}\ (\textsc{StepCaseR})$$

*Some parts of these notes are derived from notes by David Walker at Princeton.

Because we have only the non-search rules and we've made values a syntactic category rather than using the e val judgment, none of these rules have premises and authors often present them in a more compact form:

$$\begin{array}{lcl}
(\lambda x : \tau.e)\ v & \to & [v/x]e \\
\mathsf{fst}\ (v_1, v_2) & \to & v_1 \\
\mathsf{snd}\ (v_1, v_2) & \to & v_2 \\
\mathsf{case}\ \mathsf{inl}\ v\ \mathsf{of}\ \{x.e_1; y.e_2\} & \to & [v/x]e_1 \\
\mathsf{case}\ \mathsf{inr}\ v\ \mathsf{of}\ \{x.e_1; y.e_2\} & \to & [v/y]e_2
\end{array}$$

The hole-filling rules are pretty standard, so if we're defining a language this way, we might just have to define the syntax for evaluation contexts as above and give the interesting rules in that table-like form, and that's it for the dynamic semantics! You can see why this is appealing.

Note that there may be more than one way to decompose an expression, but there will be only one "useful" way (i.e., one that allows us to step with StepExp). For example,

$$\begin{array}{cl}
 & (\lambda x : \mathsf{int} \times \mathsf{int}.\mathsf{fst}\ x)\ (\mathsf{fst}\ (\overline{1}, \overline{2}), \overline{3}) \\
= & \circ\ (\mathsf{fst}\ (\overline{1}, \overline{2}), \overline{3})[\lambda x : \mathsf{int} \times \mathsf{int}.\mathsf{fst}\ x] \\
= & (\lambda x : \mathsf{int} \times \mathsf{int}.\mathsf{fst}\ x)\ (\mathsf{fst}\ \circ, \overline{3})[(\overline{1}, \overline{2})] \\
= & (\lambda x : \mathsf{int} \times \mathsf{int}.\mathsf{fst}\ x)\ (\circ, \overline{3})[\mathsf{fst}\ (\overline{1}, \overline{2})]
\end{array}$$

where only the last one allows us to step with StepExp. If we take this step, the "interesting" part of the expression moves so we have to re-decompose it to take another step, and keep doing this.

$$\begin{array}{cl}
 & (\lambda x : \mathsf{int} \times \mathsf{int}.\mathsf{fst}\ x)\ (\circ, \overline{3})[\mathsf{fst}\ (\overline{1}, \overline{2})] \\
\mapsto & (\lambda x : \mathsf{int} \times \mathsf{int}.\mathsf{fst}\ x)\ (\circ, \overline{3})[\overline{1}] \\
= & (\lambda x : \mathsf{int} \times \mathsf{int}.\mathsf{fst}\ x)\ (\overline{1}, \overline{3}) \\
= & \circ[(\lambda x : \mathsf{int} \times \mathsf{int}.\mathsf{fst}\ x)\ (\overline{1}, \overline{3})] \\
\mapsto & \circ[\mathsf{fst}\ (\overline{1}, \overline{3})] \\
\mapsto & \circ[\overline{1}] \\
= & \overline{1}
\end{array}$$

Part of the added complexity comes in the fact that we now need typing rules for evaluation contexts, as well as some extra lemmas. The typing judgment is $\mathcal{E} : \tau \rightsquigarrow \tau'$, meaning that $\mathcal{E}$ "takes" a τ (i.e., to fill the hole) and "returns" a τ'. Note that there's no need for a Γ context anymore: evaluation contexts always represent the full, closed program.

$$\frac{}{\circ : \tau \rightsquigarrow \tau}\ (\textsc{ETypeHole}) \qquad \frac{\mathcal{E} : \tau \rightsquigarrow \tau_1 \to \tau_2 \qquad \bullet \vdash e : \tau_1}{\mathcal{E}\ e : \tau \rightsquigarrow \tau_2}\ (\textsc{ETypeAppLeft})$$

$$\frac{\bullet \vdash v : \tau_1 \to \tau_2 \qquad \mathcal{E} : \tau \rightsquigarrow \tau_1}{v\ \mathcal{E} : \tau \rightsquigarrow \tau_2}\ (\textsc{ETypeAppRight}) \qquad \frac{\mathcal{E} : \tau \rightsquigarrow \tau_1 \qquad \bullet \vdash e : \tau_2}{(\mathcal{E}, e) : \tau \rightsquigarrow \tau_1 \times \tau_2}\ (\textsc{ETypePairLeft})$$

$$\frac{\bullet \vdash e : \tau_1 \qquad \mathcal{E} : \tau \rightsquigarrow \tau_2}{(\mathcal{E}, e) : \tau \rightsquigarrow \tau_1 \times \tau_2}\ (\textsc{ETypePairRight}) \qquad \frac{\mathcal{E} : \tau \rightsquigarrow \tau_1 \times \tau_2}{\mathsf{fst}\ \mathcal{E} : \tau \rightsquigarrow \tau_1}\ (\textsc{ETypeFst})$$

$$\frac{\mathcal{E} : \tau \rightsquigarrow \tau_1 \times \tau_2}{\mathsf{snd}\ \mathcal{E} : \tau \rightsquigarrow \tau_2}\ (\textsc{ETypeSnd}) \qquad \frac{\mathcal{E} : \tau \rightsquigarrow \tau_1}{\mathsf{inl}\ \mathcal{E} : \tau \rightsquigarrow \tau_1 + \tau_2}\ (\textsc{ETypeInL}) \qquad \frac{\mathcal{E} : \tau \rightsquigarrow \tau_2}{\mathsf{inr}\ \mathcal{E} : \tau \rightsquigarrow \tau_1 + \tau_2}\ (\textsc{ETypeInR})$$

$$\frac{\mathcal{E} : \tau \rightsquigarrow \tau_1 + \tau_2 \qquad x : \tau_1 \vdash e_1 : \tau' \qquad y : \tau_2 \vdash e_2 : \tau'}{\mathsf{case}\ \mathcal{E}\ \mathsf{of}\ \{x.e_1; y.e_2\} : \tau \rightsquigarrow \tau'}\ (\textsc{ETypeCase})$$

Lemma 1. *If $\mathcal{E} : \tau \rightsquigarrow \tau'$ and $\bullet \vdash e : \tau$, then $\bullet \vdash \mathcal{E}[e] : \tau'$.*

Proof. By induction on the derivation of $\mathcal{E} : \tau \rightsquigarrow \tau'$

- ETypeHole. Then $\mathcal{E} = \circ$ and $\mathcal{E}[e] = e$ and $\tau = \tau'$, so the result is true by assumption.
- ETypeAppLeft. Then $\mathcal{E} = \mathcal{E}_0\ e_0$ and $\mathcal{E}_0 : \tau \rightsquigarrow \tau_1 \to \tau'$ and $\bullet \vdash e_0 : \tau_1$ and $\mathcal{E}[e] = (\mathcal{E}_0[e])\ e_0$. By induction, we have $\bullet \vdash \mathcal{E}_0[e] : \tau_1 \to \tau'$. Apply $\to$E.
- ETypeAppRight. Then $\mathcal{E} = v\ \mathcal{E}_0$ and $\bullet \vdash v : \tau_1 \to \tau'$ and $\mathcal{E}_0 : \tau \rightsquigarrow \tau_1$ and $\mathcal{E}[e] = v\ (\mathcal{E}_0[e])$. By induction, $\bullet \vdash \mathcal{E}_0[e] : \tau_1$. Apply $\to$E.
- ETypePairLeft. Then $\mathcal{E} = (\mathcal{E}_0, e_0)$ and $\mathcal{E}_0 : \tau \rightsquigarrow \tau_1$ and $\bullet \vdash e : \tau_2$ and $\tau' = \tau_1 \times \tau_2$ and $\mathcal{E}[e] = (\mathcal{E}_0[e], e_0)$. By induction, $\bullet \vdash \mathcal{E}_0[e] : \tau_1$. Apply $\times$I.
- ETypeFst. Then $\mathcal{E} = \mathsf{fst}\ \mathcal{E}_0$ and $\mathcal{E}_0 : \tau \rightsquigarrow \tau' \times \tau_2$ and $\mathcal{E}[e] = \mathsf{fst}\ (\mathcal{E}_0[e])$. By induction, $\bullet \vdash \mathcal{E}_0[e] : \tau' \times \tau_2$. Apply $\times I_1$
- ETypeInL. Then $\mathcal{E} = \mathsf{inl}\ \mathcal{E}_0$ and $\mathcal{E}_0 : \tau \rightsquigarrow \tau_1$ and $\tau' = \tau_1 + \tau_2$ and $\mathcal{E}[e] = \mathsf{inl}\ (\mathcal{E}_0[e])$. By induction, $\bullet \vdash \mathcal{E}_0[e] : \tau_1$. Apply $+I_1$
- ETypeCase. Then $\mathcal{E} = \mathsf{case}\ \mathcal{E}_0\ \mathsf{of}\ \{x.e_1; y.e_2\}$ and $\mathcal{E}_0 : \tau \rightsquigarrow \tau_1 + \tau_2$ and $x : \tau_1 \vdash e_1 : \tau'$ and $y : \tau_2 \vdash e_2 : \tau'$ and $\mathcal{E}[e] = \mathsf{case}\ \mathcal{E}_0[e]\ \mathsf{of}\ \{x.e_1; y.e_2\}$. By induction, $\bullet \vdash \mathcal{E}_0[e] : \tau_1 + \tau_2$. Apply $+$E.

□

Lemma 2. *If $\bullet \vdash \mathcal{E}[e] : \tau$ then there exists τ' such that $\bullet \vdash e : \tau'$ and $\mathcal{E} : \tau' \rightsquigarrow \tau$.*

Proof. By induction on the structure of $\mathcal{E}$.

- $\mathcal{E} = \circ$. Then $\mathcal{E}[e] = e$. Let $\tau' = \tau$ and apply ETypeHole.
- $\mathcal{E} = \mathcal{E}_0\ e_0$. Then $\mathcal{E}[e] = (\mathcal{E}_0[e])\ e_0$. By inversion on $\to$E, $\bullet \vdash \mathcal{E}_0[e] : \tau_1 \to \tau$ and $\bullet \vdash e_0 : \tau_1$. By induction, $\bullet \vdash e : \tau'$ and $\mathcal{E}_0 : \tau' \rightsquigarrow \tau_1 \to \tau$. Apply ETypeAppLeft.
- $\mathcal{E} = v\ \mathcal{E}_0$. Then $\mathcal{E}[e] = v\ (\mathcal{E}_0[e])$. By inversion, $\bullet \vdash v : \tau_1 \to \tau$ and $\bullet \vdash \mathcal{E}_0[e] : \tau_1$. By induction, $\bullet \vdash e : \tau'$ and $\mathcal{E}_0 : \tau' \rightsquigarrow \tau_1$. Apply ETypeAppRight.
- $\mathcal{E} = (\mathcal{E}_0, e_0)$. Then $\mathcal{E}[e] = (\mathcal{E}_0[e], e_0)$. By inversion on $\times$I, $\bullet \vdash \mathcal{E}_0[e] : \tau_1$ and $\bullet \vdash e_0 : \tau_2$ and $\tau = \tau_1 \times \tau_2$. By induction, $\bullet \vdash e : \tau'$ and $\mathcal{E}_0 : \tau' \rightsquigarrow \tau_1$. Apply ETypePairLeft.
- $\mathcal{E} = \mathsf{fst}\ \mathcal{E}_0$. Then $\mathcal{E}[e] = \mathsf{fst}\ (\mathcal{E}_0[e])$. By inversion on $\times E_1$, $\bullet \vdash \mathcal{E}_0[e] : \tau \times \tau_2$. By induction, $\bullet \vdash e : \tau'$ and $\mathcal{E}_0 : \tau' \rightsquigarrow \tau \times \tau_2$. Apply ETypeFst.
- $\mathcal{E} = \mathsf{inl}\ \mathcal{E}_0$. Then $\mathcal{E}[e] = \mathsf{inl}\ (\mathcal{E}_0[e])$. By inversion on $+I_1$, $\bullet \vdash \mathcal{E}_0[e] : \tau_1$ and $\tau = \tau_1 + \tau_2$. By induction, $\bullet \vdash e : \tau'$ and $\mathcal{E}_0 : \tau' \rightsquigarrow \tau_1$. Apply ETypeInL.
- $\mathcal{E} = \mathsf{case}\ \mathcal{E}_0\ \mathsf{of}\ \{x.e_1; y.e_2\}$. Then $\mathcal{E}[e] = \mathsf{case}\ \mathcal{E}_0[e]\ \mathsf{of}\ \{x.e_1; y.e_2\}$. By inversion on $+E$, $\bullet \vdash \mathcal{E}_0[e] : \tau_1 + \tau_2$ and $x : \tau_1 \vdash e_1 : \tau$ and $y : \tau_2 \vdash e_2 : \tau$. By induction, $\bullet \vdash e : \tau'$ and $\mathcal{E}_0 : \tau' \rightsquigarrow \tau_1 + \tau_2$. Apply ETypeCase.

□

We can now prove (the one case of!) Preservation. We use as a lemma the preservation result we've already shown for the $e \to e'$ judgment. If you were specifying a new language using evaluation contexts, you'd need to prove those cases too (but just for the rules we've shown here, not the search rules).

Lemma 3 (Preservation)**.** *If $\bullet \vdash e : \tau$ and $e \mapsto e'$ then $\bullet \vdash e' : \tau$.*

Proof. By induction on the derivation of $e \mapsto e'$.

- StepExp. Then $e = \mathcal{E}[e_0]$ and $e' = \mathcal{E}[e_0']$ and $e_0 \to e_0'$. By Lemma 2, there exists τ' such that $\bullet \vdash e_0 : \tau'$ and $\mathcal{E} : \tau' \rightsquigarrow \tau$. By the previous preservation result, $\bullet \vdash e_0' : \tau'$. By Lemma 1, $\bullet \vdash e' : \tau$.

□

To prove progress, we need to show that we can decompose an expression into an evaluation context and the expression that fills it. And in particular, that this can be done in a way that the filling expression is one that can step with the non-search rules.

Lemma 4 (Decomposition)**.** *If* $\bullet \vdash e : \tau$ *then either* e *is a value or* $e = \mathcal{E}[e_0]$ *and* e_0 *is one of the following:*

- $(\lambda x : \tau'.e_f)\ v$
- $\mathsf{fst}\ (v_1, v_2)$
- $\mathsf{snd}\ (v_1, v_2)$
- $\mathsf{case}\ \mathsf{inl}\ v\ \mathsf{of}\ \{x.e_1; y.e_2\}$
- $\mathsf{case}\ \mathsf{inr}\ v\ \mathsf{of}\ \{x.e_1; y.e_2\}$

Proof. By induction on the derivation of $\bullet \vdash e : \tau$. We consider a few cases.

- unitI. Then $e = ()$ is a value.
- $\to$I. Then $e = \lambda x : \tau'.e_0$ is a value.
- $\to$ E. Then $e = e_1\ e_2$ and $\bullet \vdash e_1 : \tau_1 \to \tau$ and $\bullet \vdash e_2 : \tau_1$. By induction, e_1 is a value or $e_1 = \mathcal{E}[e_1']$ where e_1' has one of the forms above.
  - If e_1' is a value, then by canonical forms, $e_1 = \lambda x : \tau_1.e_0$. By induction, e_2 is a value or $e_2 = \mathcal{E}[e_2']$ where e_2' has one of the forms above.
    - If e_2 is a value, then $e = \circ[(\lambda x : \tau_1.e_0)\ v]$, as required.
    - If $e_2 = \mathcal{E}[e_2']$, then $e = ((\lambda x : \tau_1.e_0)\ \mathcal{E})[e_2']$.
  - If $e_1 = \mathcal{E}[e_1']$, then $e = (\mathcal{E}\ e_2)[e_1']$.

□

Lemma 5 (Progress)**.** *If* $\bullet \vdash e : \tau$*, then* e *is a value or* $e \mapsto e'$ *for some* e'*.*

Proof. By Lemma 4, either e is a value (in which case we're done) or $e = \mathcal{E}[e_0]$ and, by inspection of the cases in the conclusion of the lemma, $e_0 \to e_0'$ for some e_0'. In this case, by StepExp, $e \mapsto \mathcal{E}[e_0']$. □