Chapter 1

Environment semantics

The operational semantics of the simply typed (or untyped) λ-calculus discussed so far hinges on substitutions in reducing such expressions as applications, case expressions, and the fixed point construct. Since the definition of a substitution $[e'/x]e$ analyzes the structure of e to find all occurrences of x, a naive implementation of substitutions can be extremely inefficient in terms of time, especially because of the potential size of e. Even worse, x may not appear at all in e, in which case all the work put into the analysis of e is wasted.

This chapter presents another form of operational semantics, called *environment semantics*, which overcomes the inefficiency of the naive implementation of substitutions. The environment semantics does not entirely eliminate the need for substitutions, but it performs substitutions only if necessary by postponing them as much as possible. The development of the environment semantics also leads to the introduction of another important concept called *closures*, which are compact representations of closed λ-abstractions (*i.e.*, those containing no free variables) generated during evaluations.

Before presenting the environment semantics, we develop a new form of judgment for "evaluating" expressions (as opposed to "reducing" expressions). In comparison with the reduction judgment, the new judgment lends itself better to explaining the key idea behind the environment semantics.

1.1 Evaluation judgment

As in Chapter **??**, we consider the fragment of the simply typed λ-calculus consisting of the boolean type and function types:

$$\begin{array}{rrcl} \text{type} & A & ::= & P \mid A \rightarrow A \\ \text{base type} & P & ::= & \mathsf{bool} \\ \text{expression} & e & ::= & x \mid \lambda x\!:\!A.\,e \mid e\;e \mid \mathsf{true} \mid \mathsf{false} \mid \mathsf{if}\;e\;\mathsf{then}\;e\;\mathsf{else}\;e \\ \text{value} & v & ::= & \lambda x\!:\!A.\,e \mid \mathsf{true} \mid \mathsf{false} \end{array}$$

In a certain sense, a reduction judgment $e \mapsto e'$ takes a single *small* step toward completing the evaluation of e, since the evaluation of e to a value requires a sequence of such steps in general. For this reason, the operational semantics based on the reduction judgment $e \mapsto e'$ is also called a *small-step* semantics.

An opposite approach is to take a single *big* step with which we immediately finish evaluating a given expression. To realize this approach, we introduce an *evaluation judgment* of the form $e \hookrightarrow v$:

$$e \hookrightarrow v \qquad \Leftrightarrow \qquad e \textit{ evaluates to } v$$

The intuition behind the evaluation judgment is that $e \hookrightarrow v$ conveys the same meaning as $e \mapsto^* v$ (which we will actually prove in Theorem 1.1). An operational semantics based on the evaluation judgment is also called a *big-step* semantics.

We refer to an inference rule deducing an evaluation judgment as an *evaluation rule*. Unlike a reduction judgment which is never applied to a value (*i.e.*, no reduction judgment of the form $v \mapsto e$), an evaluation judgment $v \hookrightarrow v$ is always valid because $v \mapsto^* v$ holds for any value v. The three reduction rules Lam, Arg, and App for applications (under the call-by-value strategy) are now merged into a single evaluation rule with three premises:

$$\frac{}{\lambda x\!:\!A.\,e \hookrightarrow \lambda x\!:\!A.\,e}\ \textbf{Lam} \qquad \frac{e_1 \hookrightarrow \lambda x\!:\!A.\,e \quad e_2 \hookrightarrow v_2 \quad [v_2/x]e \hookrightarrow v}{e_1\ e_2 \hookrightarrow v}\ \textbf{App}$$

$$\frac{}{\mathsf{true} \hookrightarrow \mathsf{true}}\ \textbf{True} \qquad \frac{}{\mathsf{false} \hookrightarrow \mathsf{false}}\ \textbf{False}$$

$$\frac{e \hookrightarrow \mathsf{true} \quad e_1 \hookrightarrow v}{\mathsf{if}\ e\ \mathsf{then}\ e_1\ \mathsf{else}\ e_2 \hookrightarrow v}\ \textbf{If}_{\mathsf{true}} \qquad \frac{e \hookrightarrow \mathsf{false} \quad e_2 \hookrightarrow v}{\mathsf{if}\ e\ \mathsf{then}\ e_1\ \mathsf{else}\ e_2 \hookrightarrow v}\ \textbf{If}_{\mathsf{false}}$$

Note that there is only one rule for each form of expression. In other words, the evaluation rules are syntax directed. Thus we may invert an evaluation rule so that its conclusion justifies the use of its premises. (See Lemma **??** for a similar example.) For example, $e_1\ e_2 \hookrightarrow v$ asserts the existence of $\lambda x\!:\!A.\,e$ and v_2 such that $e_1 \hookrightarrow \lambda x\!:\!A.\,e$, $e_2 \hookrightarrow v_2$, and $[v_2/x]e \hookrightarrow v$.

The following derivation (with evaluation rule names omitted) shows how to evaluate $(\lambda x\!:\!\mathsf{bool}.\,x)\ ((\lambda y\!:\!\mathsf{bool}.\,y)\ \mathsf{true})$ to true in a single "big" step:

$$\frac{\dfrac{}{\lambda x\!:\!\mathsf{bool}.\,x \hookrightarrow \lambda x\!:\!\mathsf{bool}.\,x} \quad \dfrac{\dfrac{}{\lambda y\!:\!\mathsf{bool}.\,y \hookrightarrow \lambda y\!:\!\mathsf{bool}.\,y} \quad \dfrac{}{\mathsf{true} \hookrightarrow \mathsf{true}} \quad \dfrac{}{[\mathsf{true}/y]y \hookrightarrow \mathsf{true}}}{(\lambda y\!:\!\mathsf{bool}.\,y)\ \mathsf{true} \hookrightarrow \mathsf{true}} \quad \dfrac{}{[\mathsf{true}/x]x \hookrightarrow \mathsf{true}}}{(\lambda x\!:\!\mathsf{bool}.\,x)\ ((\lambda y\!:\!\mathsf{bool}.\,y)\ \mathsf{true}) \hookrightarrow \mathsf{true}}$$

Theorem 1.1 states the relationship between evaluation judgments and reduction judgments; the proof consists of proofs of Propositions 1.2 and 1.3:

Theorem 1.1. *$e \hookrightarrow v$ if and only if $e \mapsto^* v$.*

Proposition 1.2. *If $e \hookrightarrow v$, then $e \mapsto^* v$.*

Proposition 1.3. *If $e \mapsto^* v$, then $e \hookrightarrow v$.*

The proof of Proposition 1.2 proceeds by rule induction on the judgment $e \hookrightarrow v$ and uses Lemma 1.4. The proof of Lemma 1.4 essentially uses mathematical induction on the length of the reduction sequence $e \mapsto^* e'$, but we recast the proof in terms of rule induction with the following inference rules:

$$\frac{}{e \mapsto^* e}\ \mathit{Refl} \qquad \frac{e \mapsto e'' \quad e'' \mapsto^* e'}{e \mapsto^* e'}\ \mathit{Trans}$$

Lemma 1.4. *Suppose $e \mapsto^* e'$.*
(1) $e\ e'' \mapsto^* e'\ e''$.
(2) $(\lambda x\!:\!A.\,e'')\ e \mapsto^* (\lambda x\!:\!A.\,e'')\ e'$.
(3) $\mathsf{if}\ e\ \mathsf{then}\ e_1\ \mathsf{else}\ e_2 \mapsto^* \mathsf{if}\ e'\ \mathsf{then}\ e_1\ \mathsf{else}\ e_2$.

Proof. By rule induction on the judgment $e \mapsto^* e'$. We consider the clause (1). The other two clauses are proven in a similar way.

Case $\dfrac{}{e \mapsto^* e}\ \mathit{Refl}$ where $e' = e$:

$e\ e'' \mapsto^* e'\ e''$ — from $e\ e'' = e'\ e''$ and the rule Refl

Case $\dfrac{e \mapsto e_t \quad e_t \mapsto^* e'}{e \mapsto^* e'}\ \mathit{Trans}$

$e_t\ e'' \mapsto^* e'\ e''$ — by induction hypothesis on $e_t \mapsto^* e'$

$e\ e'' \mapsto e_t\ e''$ from $\dfrac{e \mapsto e_t}{e\ e'' \mapsto e_t\ e''}\ \mathit{Lam}$

$e\ e'' \mapsto^* e'\ e''$ from $\dfrac{e\ e'' \mapsto e_t\ e'' \quad e_t\ e'' \mapsto^* e'\ e''}{e\ e'' \mapsto^* e'\ e''}\ \mathit{Trans}$

□

Lemma 1.5. *If $e \mapsto^* e'$ and $e' \mapsto^* e''$, then $e \mapsto^* e''$.*

Proof. By rule induction on the judgment $e \mapsto^* e'$ (not on $e' \mapsto^* e''$). □

Proof of Proposition 1.2. By rule induction on the judgment $e \hookrightarrow v$. If $e = v$, then $e \mapsto^* v$ holds by the rule *Refl*. Hence we need to consider the cases for the rules **App**, $\mathbf{If}_{\mathsf{true}}$, and $\mathbf{If}_{\mathsf{false}}$. We show the case for the rule **App**.

Case $\dfrac{e_1 \hookrightarrow \lambda x\!:\!A.\,e' \quad e_2 \hookrightarrow v_2 \quad [v_2/x]e' \hookrightarrow v}{e_1\ e_2 \hookrightarrow v}\ \mathbf{App}$ where $e = e_1\ e_2$:

$e_1 \mapsto^* \lambda x\!:\!A.\,e'$ by induction hypothesis on $e_1 \hookrightarrow \lambda x\!:\!A.\,e'$
$e_1\ e_2 \mapsto^* (\lambda x\!:\!A.\,e')\ e_2$ by Lemma 1.4
$e_2 \mapsto^* v_2$ by induction hypothesis on $e_2 \hookrightarrow v_2$
$(\lambda x\!:\!A.\,e')\ e_2 \mapsto^* (\lambda x\!:\!A.\,e')\ v_2$ by Lemma 1.4
$[v_2/x]e' \mapsto^* v$ by induction hypothesis on $[v_2/x]e' \hookrightarrow v$

$(\lambda x\!:\!A.\,e')\ v_2 \mapsto^* v$ $\dfrac{\dfrac{}{(\lambda x\!:\!A.\,e')\ v_2 \mapsto [v_2/x]e'}\ \mathit{App} \quad [v_2/x]e' \mapsto^* v}{(\lambda x\!:\!A.\,e')\ v_2 \mapsto^* v}\ \mathit{Trans}$

$e_1\ e_2 \mapsto^* v$ from Lemma 1.5 and $e_1\ e_2 \mapsto^* (\lambda x\!:\!A.\,e')\ e_2$, $(\lambda x\!:\!A.\,e')\ e_2 \mapsto^* (\lambda x\!:\!A.\,e')\ v_2$, $(\lambda x\!:\!A.\,e')\ v_2 \mapsto^* v$.

□

The proof of Proposition 1.3 proceeds by rule induction on the judgment $e \mapsto^* v$, but is not as straightforward as the proof of Proposition 1.2. Consider the case $\dfrac{e \mapsto e' \quad e' \mapsto^* v}{e \mapsto^* v}\ \mathit{Trans}$. By induction hypothesis on $e' \mapsto^* v$, we obtain $e' \hookrightarrow v$. Then we need to prove $e \hookrightarrow v$ using $e \mapsto e'$ and $e' \hookrightarrow v$, which is not addressed by the proposition being proven. Thus we are led to prove the following lemma before proving Proposition 1.3:

Lemma 1.6. *If $e \mapsto e'$ and $e' \hookrightarrow v$, then $e \hookrightarrow v$.*

Exercise 1.7. Prove Lemma 1.6.

Proof. By rule induction on the judgment $e \mapsto e'$ (not on $e' \hookrightarrow v$). We show a representative case:

Case $\dfrac{e_1 \mapsto e_1'}{e_1\ e_2 \mapsto e_1'\ e_2}\ \mathit{Lam}$ where $e = e_1\ e_2$ and $e' = e_1'\ e_2$:

$\dfrac{e_1' \hookrightarrow \lambda x\!:\!A.\,e_1'' \quad e_2 \hookrightarrow v_2 \quad [v_2/x]e_1'' \hookrightarrow v}{e_1'\ e_2 \hookrightarrow v}\ \mathbf{App}$

by the syntax directedness of the evaluation rules

$e_1 \hookrightarrow \lambda x\!:\!A.\,e_1''$ by induction hypothesis on $e_1 \mapsto e_1'$ with $e_1' \hookrightarrow \lambda x\!:\!A.\,e_1''$

$\dfrac{e_1 \hookrightarrow \lambda x\!:\!A.\,e_1'' \quad e_2 \hookrightarrow v_2 \quad [v_2/x]e_1'' \hookrightarrow v}{e_1\ e_2 \hookrightarrow v}\ \mathbf{App}$ □

Proof of Proposition 1.3. By rule induction on the judgment $e \mapsto^* v$.

Case $\dfrac{}{e \mapsto^* e}\ \mathit{Refl}$ where $e = v$:

$e \hookrightarrow v$ by the rule **Lam**, **True**, or **False**

Case $\dfrac{e \mapsto e' \quad e' \mapsto^* v}{e \mapsto^* v}\ \mathit{Trans}$

$e' \hookrightarrow v$ by induction hypothesis on $e' \mapsto^* v$
$e \hookrightarrow v$ by Lemma 1.6 with $e \mapsto e'$ and $e' \hookrightarrow v$

□

1.2 Environment semantics

The key idea behind the environment semantics is to postpone a substitution $[v/x]e$ in the rule **App** by storing a pair of v and x in an *environment* and then continuing to evaluate e without modifying it. When we later encounter an occurrence of x within e and need to evaluate it, we look up the environment to retrieve the actual value v for x. We use the following inductive definition of environment:

$$\mathsf{environment} \qquad \eta \quad ::= \quad \cdot \mid \eta, x \hookrightarrow v$$

$\cdot$ denotes an empty environment, and $x \hookrightarrow v$ means that variable x is to be replaced by value v. As in the definition of typing context, we assume that variables in an environment are all distinct.

We use an *environment evaluation judgment* of the form $\eta \vdash e \hookrightarrow v$:[1]

$$\eta \vdash e \hookrightarrow v \qquad \Leftrightarrow \qquad e \textit{ evaluates to } v \textit{ under environment } \eta$$

As an example, let us evaluate $[\mathsf{true}/x]\mathsf{if}\ x\ \mathsf{then}\ x\ \mathsf{else}\ x$ using the environment semantics. For the sake of simplicity, we begin with an empty environment:

$$\cdot \vdash [\mathsf{true}/x]\mathsf{if}\ x\ \mathsf{then}\ x\ \mathsf{else}\ x \hookrightarrow ?$$

Instead of applying the substitution right away, we evaluate $\mathsf{if}\ x\ \mathsf{then}\ x\ \mathsf{else}\ x$ under an augmented environment $x \hookrightarrow \mathsf{true}$ (which comes from $\cdot, x \hookrightarrow \mathsf{true}$):

$$\frac{\cdots}{x \hookrightarrow \mathsf{true} \vdash \mathsf{if}\ x\ \mathsf{then}\ x\ \mathsf{else}\ x \hookrightarrow ?}$$

To evaluate the conditional expression x, we look up the environment to retrieve its value:

$$\frac{x \hookrightarrow \mathsf{true} \vdash x \hookrightarrow \mathsf{true} \quad \cdots}{x \hookrightarrow \mathsf{true} \vdash \mathsf{if}\ x\ \mathsf{then}\ x\ \mathsf{else}\ x \hookrightarrow ?}$$

Since the conditional expression evaluates to true, we take the if branch without changing the environment:

$$\frac{x \hookrightarrow \mathsf{true} \vdash x \hookrightarrow \mathsf{true} \quad x \hookrightarrow \mathsf{true} \vdash x \hookrightarrow ?}{x \hookrightarrow \mathsf{true} \vdash \mathsf{if}\ x\ \mathsf{then}\ x\ \mathsf{else}\ x \hookrightarrow ?}$$

By looking up the environment again, we find that x evaluates to true:

$$\frac{x \hookrightarrow \mathsf{true} \vdash x \hookrightarrow \mathsf{true} \quad x \hookrightarrow \mathsf{true} \vdash x \hookrightarrow \mathsf{true}}{x \hookrightarrow \mathsf{true} \vdash \mathsf{if}\ x\ \mathsf{then}\ x\ \mathsf{else}\ x \hookrightarrow ?}$$

Now we finish evaluating $[\mathsf{true}/x]\mathsf{if}\ x\ \mathsf{then}\ x\ \mathsf{else}\ x$:

$$\frac{x \hookrightarrow \mathsf{true} \vdash x \hookrightarrow \mathsf{true} \quad x \hookrightarrow \mathsf{true} \vdash x \hookrightarrow \mathsf{true}}{x \hookrightarrow \mathsf{true} \vdash \mathsf{if}\ x\ \mathsf{then}\ x\ \mathsf{else}\ x \hookrightarrow \mathsf{true}}$$

Note that the evaluation does not even look at variable x in the else branch (because it does not need to), and thus accesses the environment only twice: one for x in the conditional expression and one for x in the

[1] Note the use of the turnstile symbol $\vdash$. Like a typing judgment $\Gamma \vdash e : A$, an environment evaluation judgment is an example of a hypothetical judgment in which $x \hookrightarrow v$ in η has exactly the same meaning as in an ordinary evaluation judgment $x \hookrightarrow v$, but is used as a hypothesis. Put another way, there is a good reason for using the syntax $x \hookrightarrow v$ for elements of environments.

if branch. In contrast, an ordinary evaluation judgment $\mathsf{if}\ x\ \mathsf{then}\ x\ \mathsf{else}\ x \hookrightarrow \mathsf{true}$ would apply a substitution $[\mathsf{true}/x]x$ three times, including the case for x in the else branch (which is unnecessary after all).

Now let us develop the rules for the environment evaluation judgment. We begin with the following (innocent-looking) set of rules:

$$\frac{x \hookrightarrow v \in \eta}{\eta \vdash x \hookrightarrow v}\ \mathbf{Var}_{\mathsf{e}} \qquad \frac{}{\eta \vdash \lambda x\!:\!A.\, e \hookrightarrow \lambda x\!:\!A.\, e}\ \mathbf{Lam}_{\mathsf{e}}$$

$$\frac{\eta \vdash e_1 \hookrightarrow \lambda x\!:\!A.\, e \quad \eta \vdash e_2 \hookrightarrow v_2 \quad \eta, x \hookrightarrow v_2 \vdash e \hookrightarrow v}{\eta \vdash e_1\ e_2 \hookrightarrow v}\ \mathbf{App}_{\mathsf{e}}$$

$$\frac{}{\eta \vdash \mathsf{true} \hookrightarrow \mathsf{true}}\ \mathbf{True}_{\mathsf{e}} \qquad \frac{}{\eta \vdash \mathsf{false} \hookrightarrow \mathsf{false}}\ \mathbf{False}_{\mathsf{e}}$$

$$\frac{\eta \vdash e \hookrightarrow \mathsf{true} \quad \eta \vdash e_1 \hookrightarrow v}{\eta \vdash \mathsf{if}\ e\ \mathsf{then}\ e_1\ \mathsf{else}\ e_2 \hookrightarrow v}\ \mathbf{If}_{\mathsf{truee}} \qquad \frac{\eta \vdash e \hookrightarrow \mathsf{false} \quad \eta \vdash e_2 \hookrightarrow v}{\eta \vdash \mathsf{if}\ e\ \mathsf{then}\ e_1\ \mathsf{else}\ e_2 \hookrightarrow v}\ \mathbf{If}_{\mathsf{falsee}}$$

The rule $\mathbf{Var}_{\mathsf{e}}$ accesses environment η to retrieve the value associated with variable x. The third premise of the rule $\mathbf{App}_{\mathsf{e}}$ augments environment η with $x \hookrightarrow v_2$ before starting to evaluating expression e.

It turns out, however, that two of these rules are faulty! (Which ones?) In order to identify the source of the problem, let us evaluate

$$(\lambda x\!:\!\mathsf{bool}.\, \lambda y\!:\!\mathsf{bool}.\, \mathsf{if}\ x\ \mathsf{then}\ y\ \mathsf{else}\ \mathsf{false})\ \mathsf{true}$$

using the environment semantics. The result must be the same closed λ-abstraction that the following evaluation judgment yields:

$$(\lambda x\!:\!\mathsf{bool}.\, \lambda y\!:\!\mathsf{bool}.\, \mathsf{if}\ x\ \mathsf{then}\ y\ \mathsf{else}\ \mathsf{false})\ \mathsf{true} \hookrightarrow \lambda y\!:\!\mathsf{bool}.\, \mathsf{if}\ \mathsf{true}\ \mathsf{then}\ y\ \mathsf{else}\ \mathsf{false}$$

To simplify the presentation, let us instead evaluate $f\ \mathsf{true}$ under an environment $\eta = f \hookrightarrow \lambda x\!:\!\mathsf{bool}.\, \lambda y\!:\!\mathsf{bool}.\, \mathsf{if}\ x\ \mathsf{then}\ y\ \mathsf{else}\ \mathsf{fal}$ which effectively evaluates the same expression. Then we expect that the following judgment holds:

$$\eta \vdash f\ \mathsf{true} \hookrightarrow \lambda y\!:\!\mathsf{bool}.\, \mathsf{if}\ \mathsf{true}\ \mathsf{then}\ y\ \mathsf{else}\ \mathsf{false}$$

The judgment, however, does not hold because $f\ \mathsf{true}$ evaluates to a λ-abstraction with a free variable x in it:

$$\frac{\begin{array}{l}\eta \vdash f \hookrightarrow \lambda x\!:\!\mathsf{bool}.\, \lambda y\!:\!\mathsf{bool}.\, \mathsf{if}\ x\ \mathsf{then}\ y\ \mathsf{else}\ \mathsf{false} \\ \eta \vdash \mathsf{true} \hookrightarrow \mathsf{true} \\ \eta, x \hookrightarrow \mathsf{true} \vdash \lambda y\!:\!\mathsf{bool}.\, \mathsf{if}\ x\ \mathsf{then}\ y\ \mathsf{else}\ \mathsf{false} \hookrightarrow \lambda y\!:\!\mathsf{bool}.\, \mathsf{if}\ x\ \mathsf{then}\ y\ \mathsf{else}\ \mathsf{false}\end{array}}{\eta \vdash f\ \mathsf{true} \hookrightarrow \lambda y\!:\!\mathsf{bool}.\, \mathsf{if}\ x\ \mathsf{then}\ y\ \mathsf{else}\ \mathsf{false}}\ \mathbf{App}_{\mathsf{e}}$$

Why does the resultant λ-abstraction contain a free variable x in it? The reason is that the rule $\mathbf{Lam}_{\mathsf{e}}$ (which is used by the third premise in the above derivation) fails to take into account the fact that values for all free variables in $\lambda x\!:\!A.\, e$ are stored in a given environment. Thus the result of evaluating $\lambda x\!:\!A.\, e$ under environment η should be not just $\lambda x\!:\!A.\, e$, but $\lambda x\!:\!A.\, e$ *together with* additional information on values for free variables in $\lambda x\!:\!A.\, e$, which is precisely the environment η itself! We write a pair of $\lambda x\!:\!A.\, e$ and η as $[\eta, \lambda x\!:\!A.\, e]$, which is called a closure because the presence of η turns $\lambda x\!:\!A.\, e$ into a closed expression. Accordingly we redefine the set of values and fix the rule $\mathbf{Lam}_{\mathsf{e}}$ as follows:

$$\begin{array}{lrcl}\mathsf{value} & v & ::= & [\eta, \lambda x\!:\!A.\, e] \mid \mathsf{true} \mid \mathsf{false}\end{array}$$

$$\frac{}{\eta \vdash \lambda x\!:\!A.\, e \hookrightarrow [\eta, \lambda x\!:\!A.\, e]}\ \mathbf{Lam}_{\mathsf{e}}$$

Now values are always closed. Note that e and v in $\eta \vdash e \hookrightarrow v$ no longer belong to the same syntactic category, since v may contain closures. That is, a value v as defined above is not necessarily an expression. In contrast, e and v in $e \hookrightarrow v$ belong to the same syntactic category, namely expressions, since neither e nor v contains closures.

Now that its first premise yields a closure, the rule $\mathbf{App}_{\mathsf{e}}$ also needs to be fixed. Suppose that e_1 evaluates to $[\eta', \lambda x\!:\!A.\, e]$ and e_2 to v_2. Since η' contains values for all free variables in $\lambda x\!:\!A.\, e$, we augment η' with $x \hookrightarrow v_2$ to obtain an environment containing values for all free variables in e. Thus we evaluate e under $\eta', x \hookrightarrow v_2$:

$$\frac{\eta \vdash e_1 \hookrightarrow [\eta', \lambda x\!:\!A.\, e] \quad \eta \vdash e_2 \hookrightarrow v_2 \quad \eta', x \hookrightarrow v_2 \vdash e \hookrightarrow v}{\eta \vdash e_1\; e_2 \hookrightarrow v}\ \mathbf{App}_{\mathsf{e}}$$

Note that the environment η under which $\lambda x\!:\!A.\, e$ is obtained is not used in evaluating e.

With the new definition of the rules $\mathbf{Lam}_{\mathsf{e}}$ and $\mathbf{App}_{\mathsf{e}}$, f true evaluates to a closure equivalent to $\lambda y\!:\!\mathsf{bool}.\ \mathsf{if}\ \mathsf{true}\ \mathsf{then}\ y\ \mathsf{else}\ \mathsf{f}$
The following derivation uses an environment $\eta = f \hookrightarrow [\cdot, \lambda x\!:\!\mathsf{bool}.\, \lambda y\!:\!\mathsf{bool}.\ \mathsf{if}\ x\ \mathsf{then}\ y\ \mathsf{else}\ \mathsf{false}]$.

$$\frac{\begin{array}{l}\eta \vdash f \hookrightarrow [\cdot, \lambda x\!:\!\mathsf{bool}.\, \lambda y\!:\!\mathsf{bool}.\ \mathsf{if}\ x\ \mathsf{then}\ y\ \mathsf{else}\ \mathsf{false}] \\ \eta \vdash \mathsf{true} \hookrightarrow \mathsf{true} \\ x \hookrightarrow \mathsf{true} \vdash \lambda y\!:\!\mathsf{bool}.\ \mathsf{if}\ x\ \mathsf{then}\ y\ \mathsf{else}\ \mathsf{false} \hookrightarrow [x \hookrightarrow \mathsf{true}, \lambda y\!:\!\mathsf{bool}.\ \mathsf{if}\ x\ \mathsf{then}\ y\ \mathsf{else}\ \mathsf{false}]\end{array}}{\eta \vdash f\ \mathsf{true} \hookrightarrow [x \hookrightarrow \mathsf{true}, \lambda y\!:\!\mathsf{bool}.\ \mathsf{if}\ x\ \mathsf{then}\ y\ \mathsf{else}\ \mathsf{false}]}\ \mathbf{App}_{\mathsf{e}}$$

In order to show the correctness of the environment semantics, we define two mappings: $e \mathbin{@} \eta$ which replaces each free variable in e by a closed value, and $v^\star$ which expands closures in v into closed λ-abstractions:

$$\begin{array}{rcl} e \mathbin{@} \cdot & = & e \\ e \mathbin{@} \eta, x \hookrightarrow v & = & [v^\star/x](e \mathbin{@} \eta) \end{array} \qquad \begin{array}{rcl} [\eta, \lambda x\!:\!A.\, e]^\star & = & (\lambda x\!:\!A.\, e) \mathbin{@} \eta \\ \mathsf{true}^\star & = & \mathsf{true} \\ \mathsf{false}^\star & = & \mathsf{false} \end{array}$$

The following propositions state the correctness of the environment semantics:

Proposition 1.8. *If $\eta \vdash e \hookrightarrow v$, then $e \mathbin{@} \eta \hookrightarrow v^\star$.*

Proposition 1.9. *If $e \hookrightarrow v$, then $\cdot \vdash e \hookrightarrow v'$ and $v'^\star = v$.*

In order to simplify their proofs, we introduce an equivalence relation $\equiv_{\mathsf{c}}$:

Definition 1.10.
$v \equiv_{\mathsf{c}} v'$ if and only if $v^\star = v'^\star$.
$\eta \equiv_{\mathsf{c}} \eta'$ if and only if $x \hookrightarrow v \in \eta$ means $x \hookrightarrow v' \in \eta'$ such that $v \equiv_{\mathsf{c}} v'$, and vice versa.

Intuitively $v \equiv_{\mathsf{c}} v'$ means that v and v' (which may contain closures) represent the same value in the simply typed λ-calculus.

Lemma 1.11.

$$\begin{array}{rcl} (e\; e') \mathbin{@} \eta & = & (e \mathbin{@} \eta)\; (e' \mathbin{@} \eta) \\ (\lambda x\!:\!A.\, e) \mathbin{@} \eta & = & \lambda x\!:\!A.\, (e \mathbin{@} \eta) \\ (\mathsf{if}\ e\ \mathsf{then}\ e_1\ \mathsf{else}\ e_2) \mathbin{@} \eta & = & \mathsf{if}\ e \mathbin{@} \eta\ \mathsf{then}\ e_1 \mathbin{@} \eta\ \mathsf{else}\ e_2 \mathbin{@} \eta \end{array}$$

Proof of Proposition 1.8. By rule induction on the judgment $\eta \vdash e \hookrightarrow v$. □

Lemma 1.12. *If $\eta \vdash e \hookrightarrow v$ and $\eta \equiv_{\mathsf{c}} \eta'$, then $\eta' \vdash e \hookrightarrow v'$ and $v \equiv_{\mathsf{c}} v'$.*

Lemma 1.13. *If $\eta \vdash [v/x]e \hookrightarrow v'$, then $\eta, x \hookrightarrow v \vdash e \hookrightarrow v''$ and $v' \equiv_{\mathsf{c}} v''$.*

Lemma 1.14. *If $\eta \vdash e \mathbin{@} \eta' \hookrightarrow v$, then $\eta, \eta' \vdash e \hookrightarrow v'$ and $v \equiv_{\mathsf{c}} v'$.*

Proof of Proposition 1.9. By rule induction on the judgment $e \hookrightarrow v$. □

1.3 Abstract machine E

The environment evaluation judgment $\eta \vdash e \hookrightarrow v$ exploits environments and closures to dispense with substitutions when evaluating expressions. Still, however, it is not suitable for a practical implementation of the operational semantics because a single judgment $\eta \vdash e \hookrightarrow v$ accounts for the entire evaluation of a given expression. This section develops an abstract machine E which, like the abstract machine C, is based on a reduction judgment (derived from the environment evaluation judgment), and, unlike the abstract machine C, makes no use of substitutions.

As in the abstract machine C, there are two kinds of states in the abstract machine E. The key difference is that the state analyzing a given expression now requires an environment; the definition of stack is also slightly different because of the use of environments:

- $\sigma \blacktriangleright e \mathbin{@} \eta$ means that the machine is currently analyzing e under the environment η. In order to evaluate a variable in e, we look up the environment η.

- $\sigma \blacktriangleleft v$ means that the machine is currently returing v to the stack σ. We do not need an environment for v because the evaluation of v has been finished.

If an expression e evaluates to a value v, the initial state of the machine would be $\square \blacktriangleright e \mathbin{@} \cdot$ and the final state $\square \blacktriangleleft v$ where $\square$ denotes an empty stack.

The formal definition of the abstract machine E is given as follows:

$$
\begin{array}{rrcl}
\textsf{value} & v & ::= & [\eta, \lambda x\!:\!A.\, e] \mid \textsf{true} \mid \textsf{false} \\
\textsf{environment} & \eta & ::= & \cdot \mid \eta, x \hookrightarrow v \\
\textsf{frame} & \phi & ::= & \square_\eta\ e \mid [\eta, \lambda x\!:\!A.\, e]\ \square \mid \textsf{if}\ \square_\eta\ \textsf{then}\ e_1\ \textsf{else}\ e_2 \\
\textsf{stack} & \sigma & ::= & \square \mid \sigma; \phi \\
\textsf{state} & s & ::= & \sigma \blacktriangleright e \mathbin{@} \eta \mid \sigma \blacktriangleleft v
\end{array}
$$

An important difference from the abstract machine C is that a hole within a frame may now need an environment:

- A frame $\square_\eta\ e$ indicates that an application $e'\ e$ is being reduced and that the environment under which to evaluate $e'\ e$ is η. Hence, after finishing the reduction of e', we reduce e under environment η.

- A frame $\textsf{if}\ \square_\eta\ \textsf{then}\ e_1\ \textsf{else}\ e_2$ indicates that a conditional construct $\textsf{if}\ e\ \textsf{then}\ e_1\ \textsf{else}\ e_2$ is being reduced and that the environment under which to evaluate $\textsf{if}\ e\ \textsf{then}\ e_1\ \textsf{else}\ e_2$ is η. Hence, after finishing the reduction of e, we reduce either e_1 or e_2 (depending on the result of reducing e) under environment η.

Then why do we not need an environment in a frame $[\eta, \lambda x\!:\!A.\, e]\ \square$? Recall from the rule $\mathbf{App}_{\mathsf{e}}$ that after evaluating e_1 to $[\eta, \lambda x\!:\!A.\, e]$ and e_2 to v_2, we evaluate e under an environment $\eta, x \hookrightarrow v_2$. Thus η inside the closure $[\eta, \lambda x\!:\!A.\, e]$ is the environment to be used after finishing the reduction of whatever expression is to fill the hole $\square$, and there is no need to annotate $\square$ with another environment.

With this intuition in mind, we are now ready to develop the reduction rules for the abstract machine E. We use a reduction judgment $s \mapsto_{\mathsf{E}} s'$ for a state transition; we write $\mapsto_{\mathsf{E}}^*$ for the reflexive and transitive closure of $\mapsto_{\mathsf{E}}$. Pay close attention to the use of an environment $\eta, x \hookrightarrow v$ in the rule App_{E}.

$$\frac{x \hookrightarrow v \in \eta}{\sigma \blacktriangleright x \mathbin{@} \eta \mapsto_{\mathsf{E}} \sigma \blacktriangleleft v}\ Var_{\mathsf{E}}$$

$$\frac{}{\sigma \blacktriangleright \lambda x\!:\!A.\, e \mathbin{@} \eta \mapsto_{\mathsf{E}} \sigma \blacktriangleleft [\eta, \lambda x\!:\!A.\, e]}\ Closure_{\mathsf{E}}$$

$$\frac{}{\sigma \blacktriangleright e_1\ e_2 \mathbin{@} \eta \mapsto_{\mathsf{E}} \sigma; \square_\eta\ e_2 \blacktriangleright e_1 \mathbin{@} \eta}\ Lam_{\mathsf{E}}$$

$$\frac{}{\sigma; \square_\eta\ e_2 \blacktriangleleft [\eta', \lambda x\!:\!A.\, e] \mapsto_{\mathsf{E}} \sigma; [\eta', \lambda x\!:\!A.\, e]\ \square \blacktriangleright e_2 \mathbin{@} \eta}\ Arg_{\mathsf{E}}$$

$$\frac{}{\sigma; [\eta, \lambda x \colon A.\, e]\ \Box \blacktriangleleft v \mapsto_{\mathsf{E}} \sigma \blacktriangleright e \ @\ \eta, x \hookrightarrow v}\ App_{\mathsf{E}}$$

$$\frac{}{\sigma \blacktriangleright \mathsf{true}\ @\ \eta \mapsto_{\mathsf{E}} \sigma \blacktriangleleft \mathsf{true}}\ True_{\mathsf{E}} \quad \frac{}{\sigma \blacktriangleright \mathsf{false}\ @\ \eta \mapsto_{\mathsf{E}} \sigma \blacktriangleleft \mathsf{false}}\ False_{\mathsf{E}}$$

$$\frac{}{\sigma \blacktriangleright \mathsf{if}\ e\ \mathsf{then}\ e_1\ \mathsf{else}\ e_2\ @\ \eta \mapsto_{\mathsf{E}} \sigma; \mathsf{if}\ \Box_\eta\ \mathsf{then}\ e_1\ \mathsf{else}\ e_2 \blacktriangleright e\ @\ \eta}\ If_{\mathsf{E}}$$

$$\frac{}{\sigma; \mathsf{if}\ \Box_\eta\ \mathsf{then}\ e_1\ \mathsf{else}\ e_2 \blacktriangleleft \mathsf{true} \mapsto_{\mathsf{E}} \sigma \blacktriangleright e_1\ @\ \eta}\ If_{\mathsf{trueE}}$$

$$\frac{}{\sigma; \mathsf{if}\ \Box_\eta\ \mathsf{then}\ e_1\ \mathsf{else}\ e_2 \blacktriangleleft \mathsf{false} \mapsto_{\mathsf{E}} \sigma \blacktriangleright e_2\ @\ \eta}\ If_{\mathsf{falseE}}$$

An example of a reduction sequence is shown below:

$$\begin{array}{lll}
 & \Box \blacktriangleright (\lambda x \colon \mathsf{bool}.\, \lambda y \colon \mathsf{bool}.\, \mathsf{if}\ x\ \mathsf{then}\ y\ \mathsf{else}\ \mathsf{false})\ \mathsf{true}\ \mathsf{true}\ @\ \cdot & Lam_{\mathsf{E}} \\
\mapsto_{\mathsf{E}} & \Box; \Box.\ \mathsf{true} \blacktriangleright (\lambda x \colon \mathsf{bool}.\, \lambda y \colon \mathsf{bool}.\, \mathsf{if}\ x\ \mathsf{then}\ y\ \mathsf{else}\ \mathsf{false})\ \mathsf{true}\ @\ \cdot & Lam_{\mathsf{E}} \\
\mapsto_{\mathsf{E}} & \Box; \Box.\ \mathsf{true}; \Box.\ \mathsf{true} \blacktriangleright \lambda x \colon \mathsf{bool}.\, \lambda y \colon \mathsf{bool}.\, \mathsf{if}\ x\ \mathsf{then}\ y\ \mathsf{else}\ \mathsf{false}\ @\ \cdot & Closure_{\mathsf{E}} \\
\mapsto_{\mathsf{E}} & \Box; \Box.\ \mathsf{true}; \Box.\ \mathsf{true} \blacktriangleleft [\cdot, \lambda x \colon \mathsf{bool}.\, \lambda y \colon \mathsf{bool}.\, \mathsf{if}\ x\ \mathsf{then}\ y\ \mathsf{else}\ \mathsf{false}] & Arg_{\mathsf{E}} \\
\mapsto_{\mathsf{E}} & \Box; \Box.\ \mathsf{true}; [\cdot, \lambda x \colon \mathsf{bool}.\, \lambda y \colon \mathsf{bool}.\, \mathsf{if}\ x\ \mathsf{then}\ y\ \mathsf{else}\ \mathsf{false}]\ \Box \blacktriangleright \mathsf{true}\ @\ \cdot & True_{\mathsf{E}} \\
\mapsto_{\mathsf{E}} & \Box; \Box.\ \mathsf{true}; [\cdot, \lambda x \colon \mathsf{bool}.\, \lambda y \colon \mathsf{bool}.\, \mathsf{if}\ x\ \mathsf{then}\ y\ \mathsf{else}\ \mathsf{false}]\ \Box \blacktriangleleft \mathsf{true} & App_{\mathsf{E}} \\
\mapsto_{\mathsf{E}} & \Box; \Box.\ \mathsf{true} \blacktriangleright \lambda y \colon \mathsf{bool}.\, \mathsf{if}\ x\ \mathsf{then}\ y\ \mathsf{else}\ \mathsf{false}\ @\ x \hookrightarrow \mathsf{true} & Closure_{\mathsf{E}} \\
\mapsto_{\mathsf{E}} & \Box; \Box.\ \mathsf{true} \blacktriangleleft [x \hookrightarrow \mathsf{true}, \lambda y \colon \mathsf{bool}.\, \mathsf{if}\ x\ \mathsf{then}\ y\ \mathsf{else}\ \mathsf{false}] & Arg_{\mathsf{E}} \\
\mapsto_{\mathsf{E}} & \Box; [x \hookrightarrow \mathsf{true}, \lambda y \colon \mathsf{bool}.\, \mathsf{if}\ x\ \mathsf{then}\ y\ \mathsf{else}\ \mathsf{false}]\ \Box \blacktriangleright \mathsf{true}\ @\ \cdot & True_{\mathsf{E}} \\
\mapsto_{\mathsf{E}} & \Box; [x \hookrightarrow \mathsf{true}, \lambda y \colon \mathsf{bool}.\, \mathsf{if}\ x\ \mathsf{then}\ y\ \mathsf{else}\ \mathsf{false}]\ \Box \blacktriangleleft \mathsf{true} & App_{\mathsf{E}} \\
\mapsto_{\mathsf{E}} & \Box \blacktriangleright \mathsf{if}\ x\ \mathsf{then}\ y\ \mathsf{else}\ \mathsf{false}\ @\ x \hookrightarrow \mathsf{true}, y \hookrightarrow \mathsf{true} & If_{\mathsf{E}} \\
\mapsto_{\mathsf{E}} & \Box; \mathsf{if}\ \Box_{x \hookrightarrow \mathsf{true}, y \hookrightarrow \mathsf{true}}\ \mathsf{then}\ y\ \mathsf{else}\ \mathsf{false} \blacktriangleright x\ @\ x \hookrightarrow \mathsf{true}, y \hookrightarrow \mathsf{true} & Var_{\mathsf{E}} \\
\mapsto_{\mathsf{E}} & \Box; \mathsf{if}\ \Box_{x \hookrightarrow \mathsf{true}, y \hookrightarrow \mathsf{true}}\ \mathsf{then}\ y\ \mathsf{else}\ \mathsf{false} \blacktriangleleft \mathsf{true} & If_{\mathsf{trueE}} \\
\mapsto_{\mathsf{E}} & \Box \blacktriangleright y\ @\ x \hookrightarrow \mathsf{true}, y \hookrightarrow \mathsf{true} & Var_{\mathsf{E}} \\
\mapsto_{\mathsf{E}} & \Box \blacktriangleleft \mathsf{true} &
\end{array}$$

The correctness of the abstract machine E is stated as follows:

Theorem 1.15. *$\eta \vdash e \hookrightarrow v$ if and only if $\sigma \blacktriangleright e\ @\ \eta \mapsto^*_{\mathsf{E}} \sigma \blacktriangleleft v$.*

1.4 Fixed point construct in the abstract machine E

In Section **??**, we have seen that a typical functional language based on the call-by-value strategy requires that e in $\mathsf{fix}\ x \colon A.\, e$ be a λ-abstraction. In extending the abstraction machine E with the fixed point construct, it is mandatory that e in $\mathsf{fix}\ x \colon A.\, e$ be a value (although values other than λ-abstractions or their pairs/tuples for e would not be particularly useful).

Recall the reduction rule for the fixed point construct:

$$\mathsf{fix}\ x \colon A.\, e \mapsto [\mathsf{fix}\ x \colon A.\, e/x]e$$

Since the abstract machine E does not use substitutions, a reduction of $\mathsf{fix}\ x \colon A.\, e$ must store $x \hookrightarrow \mathsf{fix}\ x \colon A.\, e$ in an environment. Thus we could consider the following reduction rule to incorporate the fixed point construct:

$$\frac{}{\sigma \blacktriangleright \mathsf{fix}\ x \colon A.\, e\ @\ \eta \mapsto_{\mathsf{E}} \sigma \blacktriangleright e\ @\ \eta, x \hookrightarrow \mathsf{fix}\ x \colon A.\, e}\ Fix_{\mathsf{E}}$$

Unfortunately the rule Fix_{E} violates the invariant that an environment associates variables with *values* rather than with general expressions. Since $\mathsf{fix}\ x \colon A.\, e$ is not a value, $x \hookrightarrow \mathsf{fix}\ x \colon A.\, e$ cannot be a valid element of an environment.

Thus we are led to restrict the fixed point construct to λ-abstractions only. In other words, we consider the fixed point construct of the form $\mathsf{fix}\ f \colon A \to B.\, \lambda x \colon A.\, e$ only. (We use the same idea to allow pairs/tuples of λ-abstractions in the fixed point construct.) Moreover we write $\mathsf{fix}\ f \colon A \to B.\, \lambda x \colon A.\, e$ as $\mathsf{fun}\ f\ x \colon A.\, e$ and *regard it as a value.* Then $\mathsf{fun}\ f\ x \colon A.\, e$ may be interpreted as follows:

- $\mathsf{fun}\ f\ x\!:\!A.\,e$ denotes a recursive function f with a formal argument x of type A and a body e.

Since $\mathsf{fun}\ f\ x\!:\!A.\,e$ denotes a recursive function, e may contain references to f.

The abstract syntax for the abstract machine E now allows $\mathsf{fun}\ f\ x\!:\!A.\,e$ as an expression and $[\eta, \mathsf{fun}\ f\ x\!:\!A.\,e]$ as a new form of closure:

$$
\begin{array}{rrcl}
\text{expression} & e & ::= & \cdots \mid \mathsf{fun}\ f\ x\!:\!A.\,e \\
\text{value} & v & ::= & \cdots \mid [\eta, \mathsf{fun}\ f\ x\!:\!A.\,e] \\
\text{frame} & \phi & ::= & \cdots \mid [\eta, \mathsf{fun}\ f\ x\!:\!A.\,e]\ \square
\end{array}
$$

A typing rule for $\mathsf{fun}\ f\ x\!:\!A.\,e$ may be obtained as an instance of the rule Fix, but it is also instructive to directly derive the rule according to the interpretation of $\mathsf{fun}\ f\ x\!:\!A.\,e$. Since e may contain references to both f (because f is a recursive function) and x (because x is a formal argument), the typing context for e contains type bindings for both f and x:

$$
\frac{\Gamma, f : A \to B, x : A \vdash e : B}{\Gamma \vdash \mathsf{fun}\ f\ x\!:\!A.\,e : A \to B}\ \mathsf{Fun}
$$

The reduction rules for $\mathsf{fun}\ f\ x\!:\!A.\,e$ are similar to those for λ-abstractions, except that the rule App^R_{E} augments the environment with not only $x \hookrightarrow v$ but also $f \hookrightarrow [\eta, \mathsf{fun}\ f\ x\!:\!A.\,e]$ because f is a recursive function:

$$
\frac{}{\sigma \blacktriangleright \mathsf{fun}\ f\ x\!:\!A.\,e\ @\ \eta \mapsto_{\mathsf{E}} \sigma \blacktriangleleft [\eta, \mathsf{fun}\ f\ x\!:\!A.\,e]}\ Closure^R_{\mathsf{E}}
$$

$$
\frac{}{\sigma; \square_\eta\ e_2 \blacktriangleleft [\eta', \mathsf{fun}\ f\ x\!:\!A.\,e] \mapsto_{\mathsf{E}} \sigma; [\eta', \mathsf{fun}\ f\ x\!:\!A.\,e]\ \square \blacktriangleright e_2\ @\ \eta}\ Arg^R_{\mathsf{E}}
$$

$$
\frac{}{\sigma; [\eta, \mathsf{fun}\ f\ x\!:\!A.\,e]\ \square \blacktriangleleft v \mapsto_{\mathsf{E}} \sigma \blacktriangleright e\ @\ \eta, f \hookrightarrow [\eta, \mathsf{fun}\ f\ x\!:\!A.\,e], x \hookrightarrow v}\ App^R_{\mathsf{E}}
$$

1.5 Exercises

Exercise 1.16. Why is it not a good idea to use an environment semantics based on reductions? That is, what is the problem with using a judgment of the form $\eta \vdash e \mapsto e'$?

Exercise 1.17. Extend the abstract machine E for product types and sum types.