Evaluation with environments

Jonathan Lam

2022/01/11

1 Motivation

Evaluation with substitution is not efficient because it forces the re-evaluation of the substituted expression every time it is encountered. A more efficient involves an environment model, where variable values are evaluated and stored in an environment when bound and looked-up when encountered.

(Is evaluation with substitution considered normal order evaluation? This seems similar to normal/applicative order evaluation described in SICP 1.1.5.)

2 Overview

The irreducible judgment (for internal expressions) in Hazel is not d val, but rather $E \vdash d$ final. Thus, final expressions evaluate to themselves. Variables evaluate to the final value that they are bound to (assuming they are bound; otherwise they are free and thus final). Lambdas evaluate to a closure type. The evaluation of a let-expression or function application extends the current environment with the newly-bound variable. For function applications, the current environment is first extended with the closure environment before binding the new variable. When extending an environment ($E :: E'$ or $E, x \leftarrow d$), bindings on the right overwrite bindings on the left.

The treatment of holes occurs in two places: when evaluating holes, and when evaluating lambda functions. (Previously, updating the hole environment was handled by substitution.) When evaluating an empty or nonempty hole, the hole environment should be the environment. When evaluating a lambda, not only do we wrap it in its closure with its environment, but all of the holes contained in the body expression must also set the environment to the current (incomplete) environment. This is because we want holes in function bodies to have an environment for users to inspect, even if evaluation never reaches the function body.

The following metatheorem states that environments only include final terms.

Theorem 1 *If the variable binding $x \leftarrow d$ exists in E, then d final.*

This can be proved by induction on an empty environment by observing that all terms added to an environment must be final.

Big-step semantics

The judgment rules for evaluating variables, lambdas (which evaluate to closures), function application, `let`-expressions (very similar to function application), and a sample binary operator are shown.

$\boxed{E \vdash d \Downarrow d'}$ Internal expression d evaluates to d' given environment E

$$\frac{E \vdash d \ \mathsf{final}}{E \vdash d \Downarrow d} \text{ EvalB-Final} \qquad \frac{}{E, x \leftarrow d \vdash x \Downarrow d} \text{ EvalB-Var}$$

$$\frac{[(\!|d''|\!)_\sigma / (\!|d''|\!)_{\sigma::E}][(\!|\,|\!)_\sigma / (\!|\,|\!)_{\sigma::E}]d = d'}{E \vdash (\lambda x : \tau.d) \Downarrow [E](\lambda x : \tau.d')} \text{ EvalB-Lam}$$

$$\frac{E \vdash d_1 \Downarrow d_1' \qquad d_1' \neq ([E']\lambda x : \tau.d) \qquad E \vdash d_2 \Downarrow d_2'}{E \vdash d_1(d_2) \Downarrow d_1'(d_2')} \text{ EvalB-Ap}_1$$

$$\frac{\begin{array}{c} E \vdash d_1 \Downarrow ([E']\lambda x : \tau.d_1') \\ E \vdash d_2 \Downarrow d_2' \qquad E :: E', x \leftarrow d_2' \vdash d_1' \Downarrow d \end{array}}{E \vdash d_1(d_2) \Downarrow d} \text{ EvalB-Ap}_2$$

$$\frac{E \vdash d_2 \Downarrow d_2' \qquad E, x \leftarrow d_2' \vdash d_1 \Downarrow d}{E \vdash \texttt{let}\ \ x = d_2\ \ \texttt{in}\ \ d_1 \Downarrow d} \text{ EvalB-Let}$$

$$\frac{E \vdash d_1 \Downarrow d_1' \qquad E \vdash d_2 \Downarrow d_2' \qquad (d_1 \neq \underline{n_1} \vee d_2 \neq \underline{n_2})}{E \vdash d_1 + d_2 \Downarrow d_1' + d_2'} \text{ EvalB-Op}_1$$

$$\frac{}{E \vdash (\!|\,|\!)_\sigma \Downarrow (\!|\,|\!)_{\sigma::E}} \text{ EvalB-EHole} \qquad \frac{E \vdash d \Downarrow d'}{E \vdash (\!|d|\!)_\sigma \Downarrow (\!|d'|\!)_{\sigma::E}} \text{ EvalB-NEHole}$$

$$\frac{E \vdash d_1 \Downarrow \underline{n_1} \qquad E \vdash d_2 \Downarrow \underline{n_2}}{E \vdash d_1 + d_2 \Downarrow \underline{n_1 + n_2}} \text{ EvalB-Op}_2$$

Small-step semantics

The small-step evaluation judgments equivalent to the above big-step judgments are shown below. This assumes an evaluation context $\mathcal{E}$ as described in the POPL 2019 paper, which evaluates subexpressions down to final expressions.

$\boxed{E \vdash d \to d'}$ Internal expression d takes an instruction transition to d' given environment E

$$\frac{}{E, x \leftarrow d \vdash x \to d}\ \text{EvalS-Var} \qquad \frac{[(\!|d''|\!)_\sigma/(\!|d''|\!)_{\sigma::E}][(\!|\,|\!)_\sigma/(\!|\,|\!)_{\sigma::E}]d = d'}{E \vdash (\lambda x : \tau.d) \to ([E]\lambda x : \tau.d')}\ \text{EvalS-Lam}$$

$$\frac{E \vdash d_2\ \mathsf{final} \qquad E :: E', x \leftarrow d_2 \vdash d_1 \to d_1'}{E \vdash ([E']\lambda x : \tau.d_1)(d_2) \to ([E']\lambda x : \tau.d_1')(d_2)}\ \text{EvalS-Ap}_1$$

$$\frac{E \vdash d_2\ \mathsf{final} \qquad E :: E', x \leftarrow d_2 \vdash d_1\ \mathsf{final}}{E \vdash ([E']\lambda x : \tau.d_1)(d_2) \to d_1}\ \text{EvalS-Ap}_2$$

$$\frac{E \vdash d_2\ \mathsf{final} \qquad E, x \leftarrow d_2 \vdash d_1 \to d_1'}{E \vdash \texttt{let}\ \ x = d_2\ \ \texttt{in}\ \ d_1 \to \texttt{let}\ \ x = d_2\ \ \texttt{in}\ \ d_1'}\ \text{EvalS-Let}_1$$

$$\frac{E \vdash d_2\ \mathsf{final} \qquad E, x \leftarrow d_2 \vdash d_1\ \mathsf{final}}{E \vdash \texttt{let}\ \ x = d_2\ \ \texttt{in}\ \ d_1 \to d_1}\ \text{EvalS-Let}_2$$

$$\frac{}{E \vdash (\!|\,|\!)_\sigma \to (\!|\,|\!)_{\sigma::E}}\ \text{EvalS-EHole} \qquad \frac{E \vdash d\ \mathsf{final}}{E \vdash (\!|d|\!)_\sigma \to (\!|d|\!)_{\sigma::E}}\ \text{EvalS-NEHole}$$

$$\frac{}{E \vdash \underline{n_1} + \underline{n_2} \to \underline{n_1 + n_2}}\ \text{EvalS-Op}$$