# Interoperation for Lazy and Eager Evaluation

William Faught

May 10, 2011

## Abstract

Programmers forgo existing solutions to problems in other programming languages where interoperation proves too cumbersome; they remake solutions, rather than reuse them. To facilitate reuse, interoperation must resolve incompatible programming language features transparently at the boundaries between languages. To address part of this problem, this paper presents a model of computation that resolves lazy and eager evaluation strategies. Unforced values act as thunks that are used and forced where appropriate by the languages themselves and do not require programmer forethought.

# 1 Introduction

Programmers forgo existing solutions to problems in other programming languages where software interoperation proves too cumbersome; they remake solutions, rather than reuse them. To facilitate reuse, interoperation must resolve language incompatibilities transparently. To address part of this problem, we present a model of computation that resolves lazy and eager evaluation strategies.

Matthews and Findler presented a method of safe interoperation between languages with incompatible polymorphic static and dynamic type systems [1]. We observe that their method is insufficient for safe interoperation between languages with incompatible lazy and eager evaluation strategies, then explain the underlying problem, and then finally present a method of interoperation that resolves this incompatibility.

The model of computation of Matthews and Findler comprises two eager languages based on ML and Scheme. We extend their model of computation with a third language that is based on Haskell and identical to their ML-like language, except it is lazy. We introduce lists to all three languages. Hereafter, we use the

```
sh {N} (cons (wrong N "Not a number") (nil N))          →
cons (sh N (wrong N "Not a number")) (sh {N} (nil N))   →
Error: "Not a number"
```

Figure 1: Scheme forces the conversion of list construction operands.

```
sh {N} (cons (wrong N "Not a number") (nil N))          →
cons (sh N (wrong N "Not a number")) (sh {N} (nil N))   →
cons (sh N (wrong N "Not a number")) (nil N)
```

Figure 2: Scheme does not force the conversion of list construction operands.

names of Haskell, ML, and Scheme to refer to their counterparts in our model of computation.

Unlike ML and Scheme, Haskell does not evaluate function arguments or list construction operands. These three contexts comprise the set of incompatible strictness points between Haskell and ML, and Haskell and Scheme. Since Haskell permits unused erroneous or divergent expressions in these contexts and ML and Scheme do not, there are Haskell values that have no counterpart in ML and Scheme. Attempting to convert such values to ML and Scheme forces the evaluation of such expressions and breaks the transparency of interoperation.

Figure 1 demonstrates how a straightforward introduction of Haskell to the model of Matthews and Findler breaks the transparency of interoperation when converting a list construction from Haskell to Scheme. The Haskell list construction contains an erroneous operand that Scheme forces to evaluate in the process of converting the Haskell list construction. Figure 2 demonstrates Scheme correctly deferring the evaluation of the erroneous Haskell list construction operand and producing as a result the counterpart Scheme list construction.

Moreover, since the conversion of functions from ML and Scheme to Haskell requires the application of the original function to the converted Haskell argument, ML and Scheme always force the evaluation of the converted Haskell argument, even if it is never used. The application of such converted functions effectively changes the order of evaluation of Haskell and breaks the transparency of interoperation.

Likewise, figure 3 demonstrates the conversion of a function from Haskell to Scheme. Scheme forces the evaluation of the erroneous Haskell argument in the process of applying the Scheme function, even though the Haskell argument is never used. From the perspective of the outermost Haskell application, the argument must

$(\texttt{hs } (\texttt{N} \rightarrow \texttt{N}) \ (\lambda x_S.\overline{0}))\ (\texttt{wrong N "Not a number"})$ $\rightarrow$

$(\lambda x_H : \texttt{N}.\texttt{hs N } ((\lambda x_S.\overline{0})\ (\texttt{sh N } x_H)))\ (\texttt{wrong N "Not a number"})$ $\rightarrow$

$\texttt{hs N } ((\lambda x_S.\overline{0})\ (\texttt{sh N } (\texttt{wrong N "Not a number"})))$ $\rightarrow$

**Error:** "Not a number"

Figure 3: Scheme forces the conversion of arguments.

$(\texttt{hs } (\texttt{N} \rightarrow \texttt{N}) \ (\lambda x_S.\overline{0}))\ (\texttt{wrong N "Not a number"})$ $\rightarrow$

$(\lambda x_H : \texttt{N}.\texttt{hs N } ((\lambda x_S.\overline{0})\ (\texttt{sh N } x_H)))\ (\texttt{wrong N "Not a number"})$ $\rightarrow$

$\texttt{hs N } ((\lambda x_S.\overline{0})\ (\texttt{sh N } (\texttt{wrong N "Not a number"})))$ $\rightarrow$

$\texttt{hs N } \overline{0}$ $\rightarrow$

$\overline{0}$

Figure 4: Scheme does not force the conversion of arguments.

have been used, but it was not. Figure 4 demonstrates Scheme not forcing the evaluation of the Haskell argument, which allows the Scheme function to produce a number.

It is clear that in the contexts of the incompatible strictness points in ML and Scheme, reducible expressions in Haskell boundaries must not be evaluated to preserve the transparency of interoperation.

## 2 Model of Computation

The interoperation of Haskell and ML posed another problem: the conversion of type abstractions. The application of a converted type abstraction cannot substitute the type argument into the inner language directly, since the inner language has no notion of the types of the outer language.

Discuss hs t (sh t e) -¿ e / hm

Figure ? illustrates forced and unforced values at work for the cases explained in the introduction. The reductions for lines 1-4 show that the outer Haskell argument *zeroes* is not forced by the application of the inner Scheme function. The reductions for lines 4-8 show that the conversion of *zeroes* from Haskell to Scheme did not diverge, despite *zeroes* being a list of infinite size.

**Theorem 1.** *Evaluation Strategy Preservation*
 *TODO: equality used here doesn't match term equality*

3

$e_H = \mathtt{mh}\ t_M\ t_H\ e_H = \mathtt{sh}\ t_H\ e_H$. $e_M = \mathtt{hm}\ t_H\ t_M\ e_M = \mathtt{sm}\ t_M\ e_M$. $e_S = \mathtt{hs}\ t_H\ e_S = \mathtt{ms}\ t_M\ e_S$.

*Proof.* By structural induction. $\square$

(Lump equality) Instead, conversion substitutes lumps in a boundary's inner type. The application of a converted type abstraction substitutes the type argument in the boundary's outer type. Since the natural embedding [1] requires the boundary's outer and inner types to be equal, a new equality relation called lump equality is used here that allows lumps within the boundary's inner type to match any corresponding type in the boundary's outer type.

Legends of symbol and syntax names are presented in figures 5-7; Haskell is presented in figures 8-12; ML is presented in figures 13-17; Scheme is presented in figures 18-22; the unbrand function is presented in figure 23; and the lump equality relation is presented in figure 24.

| Symbol | Name |
|--------|------|
| $b$ | Brand |
| $k$ | Conversion scheme |
| $e$ | Expression |
| $F$ | Forced evaluation context |
| $f$ | Forced value |
| $\mathtt{L}$ | Lump |
| $\doteq$ | Lump equality relation |
| $\mathscr{E}$ | Meta evaluation context |
| $\overline{n}$ | Natural number |
| $\mathtt{N}$ | Natural number |
| $\rightarrow$ | Reduction relation |
| $t$ | Type |
| $y$ | Type variable |
| $\Gamma$ | Typing environment |
| $\vdash$ | Typing relation |
| $U$ | Unforced evaluation context |
| $u$ | Unforced value |
| $x$ | Variable |

Figure 5: Symbol names

| Syntax | Name |
| --- | --- |
| $+\ e\ e$ | Addition |
| `if0` $e\ e\ e$ | Condition |
| `nil` $t$ | Empty list |
| `nil` | Empty list |
| `wrong` $t\ string$ | Error |
| `wrong` $string$ | Error |
| `fix` $e$ | Fixed-point operation |
| $\lambda x : t.e$ | Function abstraction |
| $\lambda x_S.e_S$ | Function abstraction |
| $e\ e$ | Function application |
| `hm` $t_H\ t_M\ e_M$ | Haskell-ML guard |
| `hs` $k_H\ e_S$ | Haskell-Scheme guard |
| `cons` $e\ e$ | List construction |
| `hd` $e$ | List head |
| `tl` $e$ | List tail |
| `mh` $t_M\ t_H\ e_H$ | ML-Haskell guard |
| `ms` $k_M\ e_S$ | ML-Scheme guard |
| `sh` $k_H\ e_H$ | Scheme-Haskell guard |
| `sm` $k_M\ e_M$ | Scheme-ML guard |
| $-\ e\ e$ | Subtraction |
| $\Lambda y.e$ | Type abstraction |
| $e\langle t\rangle$ | Type application |
| `fun?` $e_S$ | Value predicate |
| `list?` $e_S$ | Value predicate |
| `null?` $e$ | Value predicate |
| `num?` $e_S$ | Value predicate |

Figure 6: Syntax names

| Syntax | Name |
|---|---|
| $b \diamond t$ | Branded type |
| $\forall y.t$ | Forall |
| $\forall u.k$ | Forall |
| $t \rightarrow t$ | Function abstraction |
| $k \rightarrow k$ | Function abstraction |
| $\{t\}$ | List |
| $\{k\}$ | List |

Figure 7: Syntax names

$$
\begin{aligned}
e_H \;=\;& x_H \mid u_H \mid e_H\ e_H \mid e_H\langle t_H\rangle \mid \mathtt{fix}\ e_H \mid a\ e_H\ e_H \mid \mathtt{if0}\ e_H\ e_H\ e_H \mid c\ e_H \\
& \mathtt{null?}\ e_H \mid \mathtt{wrong}\ t_H\ string \mid \mathtt{hm}\ t_H\ t_M\ e_M \mid \mathtt{hs}\ k_H\ e_S \\
u_H \;=\;& \lambda x_H : t_H.e_H \mid \Lambda y_H.e_H \mid \overline{n} \mid \mathtt{nil}\ t_H \mid \mathtt{cons}\ e_H\ e_H \mid \mathtt{hm\ L}\ t_M\ f_M \\
& \mathtt{hs\ L}\ f_S \\
t_H \;=\;& \mathtt{L} \mid \mathtt{N} \mid y_H \mid \{t_H\} \mid t_H \to t_H \mid \forall y_H.t_H \\
k_H \;=\;& \mathtt{L} \mid \mathtt{N} \mid u_H \mid \{k_H\} \mid k_H \to k_H \mid \forall u_H.k_H \mid b \diamond t_H \\
a \;=\;& + \mid - \\
c \;=\;& \mathtt{hd} \mid \mathtt{tl} \\
F_H \;=\;& [\,]_H \mid F_H\ e_H \mid F_H\langle t_H\rangle \mid \mathtt{fix}\ F_H \mid a\ F_H\ e_H \mid a\ u_H\ F_H \\
& \mathtt{if0}\ F_H\ e_H\ e_H \mid c\ F_H \mid \mathtt{null?}\ F_H \mid \mathtt{hm}\ t_H\ t_M\ F_M \mid \mathtt{hs}\ k_H\ F_S
\end{aligned}
$$

Figure 8: Haskell syntax and evaluation contexts

$$\frac{}{\vdash_H \mathtt{L}} \quad \frac{}{\vdash_H \mathtt{N}} \quad \frac{}{\Gamma, y_H \vdash_H y_H}$$

$$\frac{\Gamma \vdash_H t_H}{\Gamma \vdash_H \{t_H\}} \quad \frac{\Gamma \vdash_H t_H \quad \Gamma \vdash_H t'_H}{\Gamma \vdash_H t_H \to t'_H} \quad \frac{\Gamma, y_H \vdash_H t_H}{\Gamma \vdash_H \forall y_H.t_H}$$

$$\frac{\Gamma \vdash_H t_H \quad \Gamma, x_H : t_H \vdash_H e_H : t'_H}{\Gamma \vdash_H (\lambda x_H : t_H.e_H) : t_H \to t'_H} \quad \frac{\Gamma, y_H \vdash_H e_H : t_H}{\Gamma \vdash_H \Lambda y_H.e_H : \forall y_H.t_H} \quad \frac{}{\vdash_H \overline{n} : \mathtt{N}}$$

$$\frac{\Gamma \vdash_H t_H :}{\Gamma \vdash_H \mathtt{nil}\ t_H : \{t_H\}} \quad \frac{\Gamma \vdash_H e_H : t_H \quad \Gamma \vdash_H e'_H : \{t_H\}}{\Gamma \vdash_H \mathtt{cons}\ e_H\ e'_H : \{t_H\}} \quad \frac{}{\Gamma, x_H : t_H \vdash_H x_H : t_H}$$

$$\frac{\Gamma \vdash_H e_H : t_H \to t'_H \quad \Gamma \vdash_H e'_H : t_H}{\Gamma \vdash_H e_H\ e'_H : t'_H} \quad \frac{\Gamma \vdash_H e_H : t_H \to t_H}{\Gamma \vdash_H \mathtt{fix}\ e_H : t_H}$$

$$\frac{\Gamma \vdash_H t_H \quad \Gamma \vdash_H e_H : \forall y_H.t'_H}{\Gamma \vdash_H e_H\langle t_H \rangle : t'_H[t_H/y_H]} \quad \frac{\Gamma \vdash_H e_H : \{t_H\}}{\Gamma \vdash_H \mathtt{hd}\ e_H : t_H} \quad \frac{\Gamma \vdash_H e_H : \{t_H\}}{\Gamma \vdash_H \mathtt{tl}\ e_H : \{t_H\}}$$

$$\frac{\Gamma \vdash_H e_H : \mathtt{N} \quad \Gamma \vdash_H e'_H : \mathtt{N}}{\Gamma \vdash_H a\ e_H\ e'_H : \mathtt{N}} \quad \frac{\Gamma \vdash_H e_H : \{t_H\}}{\Gamma \vdash_H \mathtt{null?}\ e_H : \mathtt{N}} \quad \frac{\Gamma \vdash_H \lfloor k_H \rfloor \quad \Gamma \vdash_S e_S : \mathtt{TST}}{\Gamma \vdash_H \mathtt{hs}\ k_H\ e_S : \lfloor k_H \rfloor}$$

$$\frac{\Gamma \vdash_H e_H : \mathtt{N} \quad \Gamma \vdash_H e'_H : t_H \quad \Gamma \vdash_H e''_H : t_H}{\Gamma \vdash_H \mathtt{if0}\ e_H\ e'_H\ e''_H : t_H} \quad \frac{\Gamma \vdash_H t_H}{\Gamma \vdash_H \mathtt{wrong}\ t_H\ string : t_H}$$

$$\frac{\Gamma \vdash_H t_H \quad \Gamma \vdash_M t_M \quad \Gamma \vdash_M e_M : t'_M \quad t_H \doteq t_M \quad t_M = t'_M}{\Gamma \vdash_H \mathtt{hm}\ t_H\ t_M\ e_M : t_H}$$

Figure 9: Haskell typing rules

$$\mathscr{E}[(\lambda x_H : t_H.e_H)\ e'_H]_H \rightarrow \mathscr{E}[e_H[e'_H/x_H]]$$

$$\mathscr{E}[(\Lambda y_H.e_H)\langle t_H\rangle]_H \rightarrow \mathscr{E}[e_H[b \diamond t_H/y_H]]$$

$$\mathscr{E}[\texttt{fix}\ (\lambda x_H : t_H.e_H)]_H \rightarrow \mathscr{E}[e_H[\texttt{fix}\ (\lambda x_H : t_H.e_H)/x_H]]$$

$$\mathscr{E}[+\ \overline{n}\ \overline{n'}]_H \rightarrow \mathscr{E}[\overline{n + n'}]$$

$$\mathscr{E}[-\ \overline{n}\ \overline{n'}]_H \rightarrow \mathscr{E}[\overline{max(n - n', 0)}]$$

$$\mathscr{E}[\texttt{if0}\ \overline{0}\ e_H\ e'_H]_H \rightarrow \mathscr{E}[e_H]$$

$$\mathscr{E}[\texttt{if0}\ \overline{n}\ e_H\ e'_H]_H \rightarrow \mathscr{E}[e'_H]\ (n \neq 0)$$

$$\mathscr{E}[\texttt{hd}\ (\texttt{nil}\ t_H)]_H \rightarrow \mathscr{E}[\texttt{wrong}\ t_H\ \text{``Empty list''}]$$

$$\mathscr{E}[\texttt{tl}\ (\texttt{nil}\ t_H)]_H \rightarrow \mathscr{E}[\texttt{wrong}\ \{t_H\}\ \text{``Empty list''}]$$

$$\mathscr{E}[\texttt{hd}\ (\texttt{cons}\ e_H\ e'_H)]_H \rightarrow \mathscr{E}[e_H]$$

$$\mathscr{E}[\texttt{tl}\ (\texttt{cons}\ e_H\ e'_H)]_H \rightarrow \mathscr{E}[e'_H]$$

$$\mathscr{E}[\texttt{null?}\ (\texttt{nil}\ t_H)]_H \rightarrow \mathscr{E}[\overline{0}]$$

$$\mathscr{E}[\texttt{null?}\ (\texttt{cons}\ e_H\ e'_H)]_H \rightarrow \mathscr{E}[\overline{1}]$$

$$\mathscr{E}[\texttt{wrong}\ t_H\ string]_H \rightarrow \textbf{Error:}\ string$$

Figure 10: Haskell operational semantics

$\mathscr{E}[\text{hm } t_H \text{ L } (\text{mh L } t'_H \ e_H)]_H \to \mathscr{E}[e_H] \quad (t_H = t'_H \text{ and } t_H \neq \text{L})$

$\mathscr{E}[\text{hm } t_H \text{ L } (\text{mh L } t'_H \ e_H)]_H \to \mathscr{E}[\text{wrong } t_H \text{ "Type mismatch"}]$
$\qquad (t_H \neq t'_H \text{ and } t_H \neq \text{L})$

$\mathscr{E}[\text{hm } t_H \text{ L } (\text{ms L } f_S)]_H \to \mathscr{E}[\text{wrong } t_H \text{ "Bad value"}] \quad (t_H \neq \text{L})$

$\mathscr{E}[\text{hm N N } \overline{n}]_H \to \mathscr{E}[\overline{n}]$

$\mathscr{E}[\text{hm } \{t_H\} \ \{t_M\} \ (\text{nil } t'_M)]_H \to \mathscr{E}[\text{nil } t_H]$

$\mathscr{E}[\text{hm } \{t_H\} \ \{t_M\} \ (\text{cons } u_M \ u'_M)]_H \to$
$\qquad \mathscr{E}[\text{cons } (\text{hm } t_H \ t_M \ u_M) \ (\text{hm } \{t_H\} \ \{t_M\} \ u'_M)]$

$\mathscr{E}[\text{hm } (t_H \to t'_H) \ (t_M \to t'_M) \ (\lambda x_M : t''_M.e_M)]_H \to$
$\qquad \mathscr{E}[\lambda x_H : t_H.\text{hm } t'_H \ t'_M \ ((\lambda x_M : t''_M.e_M) \ (\text{mh } t_M \ t_H \ x_H))]$

$\mathscr{E}[\text{hm } (\forall y_H.t_H) \ (\forall y_M.t_M) \ (\Lambda y'_M.e_M)]_H \to \mathscr{E}[\Lambda y_H.\text{hm } t_H \ t_M[\text{L}/y_M] \ e_M[\text{L}/y'_M]]$

Figure 11: Haskell-ML operational semantics

$\mathscr{E}[\mathtt{hs\ N}\ \overline{n}]_H \to \mathscr{E}[\overline{n}]$

$\mathscr{E}[\mathtt{hs\ N}\ f_S]_H \to \mathscr{E}[\mathtt{wrong\ N}\ \text{``Not a number''}]\ (f_S \neq \overline{n})$

$\mathscr{E}[\mathtt{hs}\ \{k_H\}\ \mathtt{nil}]_H \to \mathscr{E}[\mathtt{nil}\ \lfloor k_H \rfloor]$

$\mathscr{E}[\mathtt{hs}\ \{k_H\}\ (\mathtt{cons}\ u_S\ u'_S)]_H \to \mathscr{E}[\mathtt{cons}\ (\mathtt{hs}\ k_H\ u_S)\ (\mathtt{hs}\ \{k_H\}\ u'_S)]$

$\mathscr{E}[\mathtt{hs}\ \{k_H\}\ f_S]_H \to \mathscr{E}[\mathtt{wrong}\ \lfloor \{k_H\} \rfloor\ \text{``Not a list''}]$
    $(f_S \neq \mathtt{nil}\ \text{and}\ f_S \neq \mathtt{cons}\ u_S\ u'_S)$

$\mathscr{E}[\mathtt{hs}\ (b \diamond t_H)\ (\mathtt{sh}\ (b \diamond t_H)\ e_H)]_H \to \mathscr{E}[e_H]$

$\mathscr{E}[\mathtt{hs}\ (b \diamond t_H)\ f_S]_H \to \mathscr{E}[\mathtt{wrong}\ t_H\ \text{``Brand mismatch''}]\ (f_S \neq \mathtt{sh}\ (b \diamond t_H)\ e_H)$

$\mathscr{E}[\mathtt{hs}\ (k_H \to k'_H)\ (\lambda x_S.e_S)]_H \to \mathscr{E}[\lambda x_H : \lfloor k_H \rfloor.\mathtt{hs}\ k'_H\ ((\lambda x_S.e_S)\ (\mathtt{sh}\ k_H\ x_H))]$

$\mathscr{E}[\mathtt{hs}\ (k_H \to k'_H)\ f_S]_H \to \mathscr{E}[\mathtt{wrong}\ \lfloor k_H \to k'_H \rfloor\ \text{``Not a function''}]$
    $(f_S \neq \lambda x_S.e_S)$

$\mathscr{E}[\mathtt{hs}\ (\forall u_H.k_H)\ f_S]_H \to \mathscr{E}[\Lambda y_H.\mathtt{hs}\ k_H\ f_S]$

Figure 12: Haskell-Scheme operational semantics

$$
\begin{aligned}
e_M \;&=\; x_M \mid u_M \mid e_M\ e_M \mid e_M\langle t_M\rangle \mid \texttt{fix}\ e_M \mid a\ e_M\ e_M \mid \texttt{if0}\ e_M\ e_M\ e_M \\
&\quad \texttt{cons}\ e_M\ e_M \mid c\ e_M \mid \texttt{null?}\ e_M \mid \texttt{wrong}\ t_M\ \mathit{string} \mid \texttt{ms}\ k_M\ e_S \\
u_M \;&=\; f_M \mid \texttt{mh}\ t_M\ t_H\ e_H \\
f_M \;&=\; \lambda x_M : t_M.e_M \mid \Lambda y_M.e_M \mid \overline{n} \mid \texttt{nil}\ t_M \mid \texttt{cons}\ u_M\ u_M \mid \texttt{mh L}\ t_H\ e_H \\
&\quad \texttt{ms L}\ f_S \\
t_M \;&=\; \texttt{L} \mid \texttt{N} \mid y_M \mid \{t_M\} \mid t_M \to t_M \mid \forall y_M.t_M \\
k_M \;&=\; \texttt{L} \mid \texttt{N} \mid u_M \mid \{k_M\} \mid k_M \to k_M \mid \forall u_M.k_M \mid b \diamond t_M \\
a \;&=\; + \mid - \\
c \;&=\; \texttt{hd} \mid \texttt{tl} \\
F_M \;&=\; U_M \mid \texttt{mh}\ t_M\ t_H\ F_H \\
U_M \;&=\; [\,]_M \mid F_M\ e_M \mid f_M\ U_M \mid F_M\langle t_M\rangle \mid \texttt{fix}\ F_M \mid a\ F_M\ e_M \mid a\ f_M\ F_M \\
&\quad \texttt{if0}\ F_M\ e_M\ e_M \mid \texttt{cons}\ U_M\ e_M \mid \texttt{cons}\ u_M\ U_M \mid c\ F_M \mid \texttt{null?}\ F_M \\
&\quad \texttt{ms}\ k_M\ F_S
\end{aligned}
$$

Figure 13: ML syntax and evaluation contexts

$$\overline{\vdash_M \mathtt{L}} \quad \overline{\vdash_M \mathtt{N}} \quad \overline{\Gamma, y_M \vdash_M y_M}$$

$$\frac{\Gamma \vdash_M t_M}{\Gamma \vdash_M \{t_M\}} \quad \frac{\Gamma \vdash_M t_M \quad \Gamma \vdash_M t'_M}{\Gamma \vdash_M t_M \to t'_M} \quad \frac{\Gamma, y_M \vdash_M t_M}{\Gamma \vdash_M \forall y_M.t_M}$$

$$\frac{\Gamma \vdash_M t_M \quad \Gamma, x_M : t_M \vdash_M e_M : t'_M}{\Gamma \vdash_M (\lambda x_M : t_M.e_M) : t_M \to t'_M} \quad \frac{\Gamma, y_M \vdash_M e_M : t_M}{\Gamma \vdash_M \Lambda y_M.e_M : \forall y_M.t_M} \quad \overline{\vdash_M \overline{n} : \mathtt{N}}$$

$$\frac{\Gamma \vdash_M t_M}{\Gamma \vdash_M \mathtt{nil}\ t_M : \{t_M\}} \quad \frac{\Gamma \vdash_M e_M : t_M \quad \Gamma \vdash_M e'_M : \{t_M\}}{\Gamma \vdash_M \mathtt{cons}\ e_M\ e'_M : \{t_M\}} \quad \overline{\Gamma, x_M : t_M \vdash_M x_M : t_M}$$

$$\frac{\Gamma \vdash_M e_M : t_M \to t'_M \quad \Gamma \vdash_M e'_M : t_M}{\Gamma \vdash_H e_M\ e'_M : t'_M} \quad \frac{\Gamma \vdash_M e_M : t_M \to t_M}{\Gamma \vdash_M \mathtt{fix}\ e_M : t_M}$$

$$\frac{\Gamma \vdash_M t_M \quad \Gamma \vdash_M e_M : \forall y_M.t'_M}{\Gamma \vdash_M e_M\langle t_M\rangle : t'_M[t_M/y_M]} \quad \frac{\Gamma \vdash_M e_M : \{t_M\}}{\Gamma \vdash_M \mathtt{hd}\ e_M : t_M} \quad \frac{\Gamma \vdash_M e_M : \{t_M\}}{\Gamma \vdash_M \mathtt{tl}\ e_M : \{t_M\}}$$

$$\frac{\Gamma \vdash_M e_M : \mathtt{N} \quad \Gamma \vdash_M e'_M : \mathtt{N}}{\Gamma \vdash_M a\ e_M\ e'_M : \mathtt{N}} \quad \frac{\Gamma \vdash_M e_M : \{t_M\}}{\Gamma \vdash_M \mathtt{null?}\ e_M : \mathtt{N}} \quad \frac{\Gamma \vdash_M \lfloor k_M \rfloor \quad \Gamma \vdash_S e_S : \mathtt{TST}}{\Gamma \vdash_M \mathtt{ms}\ k_M\ e_S : \lfloor k_M \rfloor}$$

$$\frac{\Gamma \vdash_M e_M : \mathtt{N} \quad \Gamma \vdash_M e'_M : t_M \quad \Gamma \vdash_M e''_M : t_M}{\Gamma \vdash_M \mathtt{if0}\ e_M\ e'_M\ e''_M : t_M} \quad \frac{\Gamma \vdash_M t_M}{\Gamma \vdash_M \mathtt{wrong}\ t_M\ string : t_M}$$

$$\frac{\Gamma \vdash_M t_M \quad \Gamma \vdash_H t_H \quad \Gamma \vdash_H e_H : t'_H \quad t_M \doteq t_H \quad t_H = t'_H}{\Gamma \vdash_M \mathtt{mh}\ t_M\ t_H\ e_H : t_M}$$

Figure 14: ML typing rules

$$\mathscr{E}[(\lambda x_M : t_M.e_M)\ u_M]_M \to \mathscr{E}[e_M[u_M/x_M]]$$

$$\mathscr{E}[(\Lambda y_M.e_M)\langle t_M\rangle]_M \to \mathscr{E}[e_M[b \diamond t_M/y_M]]$$

$$\mathscr{E}[\mathtt{fix}\ (\lambda x_M : t_M.e_M)]_M \to \mathscr{E}[e_M[\mathtt{fix}\ (\lambda x_M : t_M.e_M)/x_M]]$$

$$\mathscr{E}[+\ \overline{n}\ \overline{n}']_M \to \mathscr{E}[\overline{n + n'}]$$

$$\mathscr{E}[-\ \overline{n}\ \overline{n}']_M \to \mathscr{E}[\overline{max(n - n', 0)}]$$

$$\mathscr{E}[\mathtt{if0}\ \overline{0}\ e_M\ e'_M]_M \to \mathscr{E}[e_M]$$

$$\mathscr{E}[\mathtt{if0}\ \overline{n}\ e_M\ e'_M]_M \to \mathscr{E}[e'_M]\ (n \neq 0)$$

$$\mathscr{E}[\mathtt{hd}\ (\mathtt{nil}\ t_M)]_M \to \mathscr{E}[\mathtt{wrong}\ t_M\ \text{``Empty list''}]$$

$$\mathscr{E}[\mathtt{tl}\ (\mathtt{nil}\ t_M)]_M \to \mathscr{E}[\mathtt{wrong}\ \{t_M\}\ \text{``Empty list''}]$$

$$\mathscr{E}[\mathtt{hd}\ (\mathtt{cons}\ u_M\ u'_M)]_M \to \mathscr{E}[u_M]$$

$$\mathscr{E}[\mathtt{tl}\ (\mathtt{cons}\ u_M\ u'_M)]_M \to \mathscr{E}[u'_M]$$

$$\mathscr{E}[\mathtt{null?}\ (\mathtt{nil}\ t_M)]_M \to \mathscr{E}[\overline{0}]$$

$$\mathscr{E}[\mathtt{null?}\ (\mathtt{cons}\ u_M\ u'_M)]_M \to \mathscr{E}[\overline{1}]$$

$$\mathscr{E}[\mathtt{wrong}\ t_M\ string]_H \to \textbf{Error:}\ string$$

Figure 15: ML operational semantics

15

$\mathscr{E}[\text{mh } t_M \text{ L } (\text{hm L } t'_M \ f_M)]_M \to \mathscr{E}[f_M] \ (t_M = t'_M \text{ and } t_M \neq \text{L})$

$\mathscr{E}[\text{mh } t_M \text{ L } (\text{hm L } t'_M \ f_M)]_M \to \mathscr{E}[\text{wrong } t_M \text{ "Type mismatch"}] \ (t_M \neq t'_M \text{ and } t_M \neq \text{L})$

$\mathscr{E}[\text{mh } t_M \text{ L } (\text{hs L } f_S)]_H \to \mathscr{E}[\text{wrong } t_M \text{ "Bad value"}] \ (t_M \neq \text{L})$

$\mathscr{E}[\text{mh N N } \overline{n}]_M \to \mathscr{E}[\overline{n}]$

$\mathscr{E}[\text{mh } \{t_M\} \ \{t_H\} \ (\text{nil } t'_H)]_M \to \mathscr{E}[\text{nil } t_M]$

$\mathscr{E}[\text{mh } \{t_M\} \ \{t_H\} \ (\text{cons } e_H \ e'_H)]_M \to \mathscr{E}[\text{cons } (\text{mh } t_M \ t_H \ e_H) \ (\text{mh } \{t_M\} \ \{t_H\} \ e'_H)]$

$\mathscr{E}[\text{mh } (t_M \to t'_M) \ (t_H \to t'_H) \ (\lambda x_H : t''_H.e_H)]_M \to$

$\qquad \mathscr{E}[\lambda x_M : t_M.\text{mh } t'_M \ t'_H \ ((\lambda x_H : t''_H.e_H) \ (\text{hm } t_H \ t_M \ x_M))]$

$\mathscr{E}[\text{mh } (\forall y_M.t_M) \ (\forall y_H.t_H) \ (\Lambda y'_H.e_H)]_M \to \mathscr{E}[\Lambda y_M.\text{mh } t_M \ t_H[\text{L}/y_H] \ e_H[\text{L}/y'_H]]$

Figure 16: ML-Haskell operational semantics

$\mathscr{E}[\text{ms N } \overline{n}]_M \rightarrow \mathscr{E}[\overline{n}]$

$\mathscr{E}[\text{ms N } f_S]_M \rightarrow \mathscr{E}[\text{wrong N "Not a number"}] \ (f_S \neq \overline{n})$

$\mathscr{E}[\text{ms } \{k_M\} \text{ nil}]_M \rightarrow \mathscr{E}[\text{nil } \lfloor k_M \rfloor]$

$\mathscr{E}[\text{ms } \{k_M\} \ (\text{cons } u_S \ u'_S)]_M \rightarrow \mathscr{E}[\text{cons } (\text{ms } k_M \ u_S) \ (\text{ms } \{k_M\} \ u'_S)]$

$\mathscr{E}[\text{ms } \{k_M\} \ f_S]_M \rightarrow \mathscr{E}[\text{wrong } \lfloor \{k_M\} \rfloor \text{ "Not a list"}]$
$\qquad (f_S \neq \text{nil and } f_S \neq \text{cons } u_S \ u'_S)$

$\mathscr{E}[\text{ms } (b \diamond t_M) \ (\text{sm } (b \diamond t_M) \ u_M)]_M \rightarrow \mathscr{E}[u_M]$

$\mathscr{E}[\text{ms } (b \diamond t_M) \ f_S]_M \rightarrow \mathscr{E}[\text{wrong } \lfloor b \diamond t_M \rfloor \text{ "Brand mismatch"}]$
$\qquad (f_S \neq \text{sm } (b \diamond t_M) \ e_M)$

$\mathscr{E}[\text{ms } (k_M \rightarrow k'_M) \ (\lambda x_S.e_S)]_M \rightarrow$
$\qquad \mathscr{E}[\lambda x_M : \lfloor k_M \rfloor.\text{ms } k'_M \ ((\lambda x_S.e_S) \ (\text{sm } k_M \ x_M))]$

$\mathscr{E}[\text{ms } (k_M \rightarrow k'_M) \ f_S]_M \rightarrow \mathscr{E}[\text{wrong } \lfloor k_M \rightarrow k'_M \rfloor \text{ "Not a function"}]$
$\qquad (f_S \neq \lambda x_S.e_S)$

$\mathscr{E}[\text{ms } (\forall u_M.k_M) \ f_S]_M \rightarrow \mathscr{E}[\Lambda y_M.\text{ms } k_M \ f_S]$

Figure 17: ML-Scheme operational semantics

$$
\begin{aligned}
e_S \quad &= \quad x_S \mid u_S \mid e_S\ e_S \mid a\ e_S\ e_S \mid p\ e_S \mid \texttt{if0}\ e_S\ e_S\ e_S \mid \texttt{cons}\ e_S\ e_S \mid c\ e_S \\
&\qquad \texttt{wrong}\ string \mid \texttt{sm}\ k_M\ e_M \\
u_S \quad &= \quad f_S \mid \texttt{sh}\ k_H\ e_H \\
f_S \quad &= \quad \lambda x_S.e_S \mid \overline{n} \mid \texttt{nil} \mid \texttt{cons}\ u_S\ u_S \mid \texttt{sh}\ (b \diamond t_H)\ e_H \mid \texttt{sm}\ (b \diamond t_M)\ f_M \\
a \quad &= \quad + \mid - \\
c \quad &= \quad \texttt{hd} \mid \texttt{tl} \\
p \quad &= \quad \texttt{fun?} \mid \texttt{list?} \mid \texttt{null?} \mid \texttt{num?} \\
F_S \quad &= \quad U_S \mid \texttt{sh}\ k_H\ F_H \\
U_S \quad &= \quad [\,]_S \mid F_S\ e_S \mid f_S\ U_S \mid a\ F_S\ e_S \mid a\ f_S\ F_S \mid p\ F_S \mid \texttt{if0}\ F_S\ e_S\ e_S \\
&\qquad \texttt{cons}\ U_S\ e_S \mid \texttt{cons}\ u_S\ U_S \mid c\ F_S \mid \texttt{sm}\ k_M\ F_M
\end{aligned}
$$

Figure 18: Scheme syntax and evaluation contexts

$$\overline{\vdash_S \mathsf{TST}}$$

$$\frac{\Gamma, x_S : \mathsf{TST} \vdash_S e_S : \mathsf{TST}}{\Gamma \vdash_S \lambda x_S.e_S : \mathsf{TST}} \qquad \overline{\vdash_S \overline{n} : \mathsf{TST}} \qquad \overline{\vdash_S \mathtt{nil} : \mathsf{TST}}$$

$$\frac{\Gamma \vdash_S e_S : \mathsf{TST} \quad \Gamma \vdash_S e'_S : \mathsf{TST}}{\Gamma \vdash_S \mathtt{cons}\ e_S\ e'_S : \mathsf{TST}} \qquad \overline{\Gamma, x_S : \mathsf{TST} \vdash_S x_S : \mathsf{TST}}$$

$$\frac{\Gamma \vdash_S e_S : \mathsf{TST} \quad \Gamma \vdash_S e'_S : \mathsf{TST}}{\Gamma \vdash_H e_S\ e'_S : \mathsf{TST}} \qquad \frac{\Gamma \vdash_S e_S : \mathsf{TST}}{\Gamma \vdash_S c\ e_S : \mathsf{TST}}$$

$$\frac{\Gamma \vdash_S e_S : \mathsf{TST} \quad \Gamma \vdash_S e'_S : \mathsf{TST}}{\Gamma \vdash_S a\ e_S\ e'_S : \mathsf{TST}} \qquad \frac{\Gamma \vdash_S e_S : \mathsf{TST}}{\Gamma \vdash_S p\ e_S : \mathsf{TST}}$$

$$\frac{\Gamma \vdash_S e_S : \mathsf{TST} \quad \Gamma \vdash_S e'_S : \mathsf{TST} \quad \Gamma \vdash_S e''_S : \mathsf{TST}}{\Gamma \vdash_S \mathtt{if0}\ e_S\ e'_S\ e''_S : \mathsf{TST}} \qquad \overline{\vdash_S \mathtt{wrong}\ string : \mathsf{TST}}$$

$$\frac{\Gamma \vdash_H \lfloor k_H \rfloor \quad \Gamma \vdash_H e_H : t_H \quad \lfloor k_H \rfloor = t_H}{\Gamma \vdash_S \mathtt{sh}\ k_H\ e_H : \mathsf{TST}} \qquad \frac{\Gamma \vdash_M \lfloor k_M \rfloor \quad \Gamma \vdash_M e_M : t_M \quad \lfloor k_M \rfloor = t_M}{\Gamma \vdash_S \mathtt{sm}\ k_M\ e_M : \mathsf{TST}}$$

Figure 19: Scheme typing rules

19

$$\mathscr{E}[(\lambda x_S.e_S)\ u_S]_S \to \mathscr{E}[e_S[u_S/x_S]]$$

$$\mathscr{E}[f_S\ u_S]_S \to \mathscr{E}[\mathtt{wrong}\ \text{``Not a function''}]\ (f_S \neq \lambda x_S.e_S)$$

$$\mathscr{E}[+\ \overline{n}\ \overline{n'}]_S \to \mathscr{E}[\overline{n+n'}]$$

$$\mathscr{E}[-\ \overline{n}\ \overline{n'}]_S \to \mathscr{E}[\overline{max(n-n',0)}]$$

$$\mathscr{E}[a\ f_S\ f'_S]_S \to \mathscr{E}[\mathtt{wrong}\ \text{``Not a number''}]\ (f_S \neq \overline{n}\ \text{or}\ f'_S \neq \overline{n})$$

$$\mathscr{E}[\mathtt{if0}\ \overline{0}\ e_S\ e'_S]_S \to \mathscr{E}[e_S]$$

$$\mathscr{E}[\mathtt{if0}\ \overline{n}\ e_S\ e'_S]_S \to \mathscr{E}[e'_S]\ (n \neq 0)$$

$$\mathscr{E}[\mathtt{if0}\ f_S\ e_S\ e'_S]_S \to \mathscr{E}[\mathtt{wrong}\ \text{``Not a number''}]\ (f_S \neq \overline{n})$$

$$\mathscr{E}[c\ \mathtt{nil}]_S \to \mathscr{E}[\mathtt{wrong}\ \text{``Empty list''}]$$

$$\mathscr{E}[\mathtt{hd}\ (\mathtt{cons}\ u_S\ u'_S)]_S \to \mathscr{E}[u_S]$$

$$\mathscr{E}[\mathtt{tl}\ (\mathtt{cons}\ u_S\ u'_S)]_S \to \mathscr{E}[u'_S]$$

$$\mathscr{E}[c\ f_S]_S \to \mathscr{E}[\mathtt{wrong}\ \text{``Not a list''}]\ (f_S \neq \mathtt{nil}\ \text{and}\ f_S \neq \mathtt{cons}\ u_S\ u'_S)$$

$$\mathscr{E}[\mathtt{fun?}\ (\lambda x_S.e_S)]_S \to \mathscr{E}[\overline{0}]$$

$$\mathscr{E}[\mathtt{fun?}\ f_S]_S \to \mathscr{E}[\overline{1}]\ (f_S \neq \lambda x_S.e_S)$$

$$\mathscr{E}[\mathtt{list?}\ \mathtt{nil}]_S \to \mathscr{E}[\overline{0}]$$

$$\mathscr{E}[\mathtt{list?}\ (\mathtt{cons}\ u_S\ u'_S)]_S \to \mathscr{E}[\overline{0}]$$

$$\mathscr{E}[\mathtt{list?}\ f_S]_S \to \mathscr{E}[\overline{1}]\ (f_S \neq \mathtt{nil}\ \text{and}\ f_S \neq \mathtt{cons}\ u_S\ u'_S)$$

$$\mathscr{E}[\mathtt{null?}\ \mathtt{nil}]_S \to \mathscr{E}[\overline{0}]$$

$$\mathscr{E}[\mathtt{null?}\ f_S]_S \to \mathscr{E}[\overline{1}]\ (f_S \neq \mathtt{nil})$$

$$\mathscr{E}[\mathtt{num?}\ \overline{n}]_S \to \mathscr{E}[\overline{0}]$$

$$\mathscr{E}[\mathtt{num?}\ f_S]_S \to \mathscr{E}[\overline{1}]\ (f_S \neq \overline{n})$$

$$\mathscr{E}[\mathtt{wrong}\ string]_S \to \textbf{Error:}\ string$$

Figure 20: Scheme operational semantics

$\mathscr{E}[\text{sh L } (\text{hm L } k_M \ f_M)]_S \rightarrow \mathscr{E}[\text{wrong "Bad value"}]$

$\mathscr{E}[\text{sh L } (\text{hs L } f_S)]_S \rightarrow \mathscr{E}[f_S]$

$\mathscr{E}[\text{sh N } \overline{n}]_S \rightarrow \mathscr{E}[\overline{n}]$

$\mathscr{E}[\text{sh } \{k_H\} \ (\text{nil } t_H)]_S \rightarrow \mathscr{E}[\text{nil}]$

$\mathscr{E}[\text{sh } \{k_H\} \ (\text{cons } e_H \ e'_H)]_S \rightarrow \mathscr{E}[\text{cons } (\text{sh } k_H \ e_H) \ (\text{sh } \{k_H\} \ e'_H)]$

$\mathscr{E}[\text{sh } (k_H \rightarrow k'_H) \ (\lambda x_H : t_H.e_H)]_S \rightarrow$

$\qquad \mathscr{E}[\lambda x_S.\text{sh } k'_H \ ((\lambda x_H : t_H.e_H) \ (\text{hs } k_H \ x_S))]$

$\mathscr{E}[\text{sh } (\forall u_H.k_H) \ (\Lambda y'_H.e_H)]_S \rightarrow \mathscr{E}[\text{sh } k_H[\text{L}/u_H] \ e_H[\text{L}/y'_H]]$

Figure 21: Scheme-Haskell operational semantics

$$\mathscr{E}[\mathtt{sm}\ \mathtt{L}\ (\mathtt{mh}\ \mathtt{L}\ k_H\ e_H)]_S \to \mathscr{E}[\mathtt{wrong}\ \text{``Bad value''}]$$

$$\mathscr{E}[\mathtt{sm}\ \mathtt{L}\ (\mathtt{ms}\ \mathtt{L}\ f_S)]_S \to \mathscr{E}[f_S]$$

$$\mathscr{E}[\mathtt{sm}\ \mathtt{N}\ \overline{n}]_S \to \mathscr{E}[\overline{n}]$$

$$\mathscr{E}[\mathtt{sm}\ \{k_M\}\ (\mathtt{nil}\ t_M)]_S \to \mathscr{E}[\mathtt{nil}]$$

$$\mathscr{E}[\mathtt{sm}\ \{k_M\}\ (\mathtt{cons}\ u_M\ u'_M)]_S \to \mathscr{E}[\mathtt{cons}\ (\mathtt{sm}\ k_M\ u_M)\ (\mathtt{sm}\ \{k_M\}\ u'_M)]$$

$$\mathscr{E}[\mathtt{sm}\ (k_M \to k'_M)\ (\lambda x_M : t_M.e_M)]_S \to$$
$$\mathscr{E}[\lambda x_S.\mathtt{sm}\ k'_M\ ((\lambda x_M : t_M.e_M)\ (\mathtt{ms}\ k_M\ x_S))]$$

$$\mathscr{E}[\mathtt{sm}\ (\forall u_M.k_M)\ (\Lambda y'_M.e_M)]_S \to \mathscr{E}[\mathtt{sm}\ k_M[\mathtt{L}/u_M]\ e_M[\mathtt{L}/y'_M]]$$

Figure 22: Scheme-ML operational semantics

$$
\begin{aligned}
\lfloor \mathtt{L} \rfloor &= \mathtt{L} \\
\lfloor \mathtt{N} \rfloor &= \mathtt{N} \\
\lfloor u_H \rfloor &= y_H \\
\lfloor u_M \rfloor &= y_M \\
\lfloor \{k_H\} \rfloor &= \{\lfloor k_H \rfloor\} \\
\lfloor \{k_M\} \rfloor &= \{\lfloor k_M \rfloor\} \\
\lfloor k_H \rightarrow k_H \rfloor &= \lfloor k_H \rfloor \rightarrow \lfloor k_H \rfloor \\
\lfloor k_M \rightarrow k_M \rfloor &= \lfloor k_M \rfloor \rightarrow \lfloor k_M \rfloor \\
\lfloor \forall u_H.k_H \rfloor &= \forall u_H.\lfloor k_H \rfloor \\
\lfloor \forall u_M.k_M \rfloor &= \forall u_M.\lfloor k_M \rfloor \\
\lfloor b \diamond t_H \rfloor &= t_H \\
\lfloor b \diamond t_M \rfloor &= t_M
\end{aligned}
$$

Figure 23: Unbrand function

$$x \doteq x$$
$$x \doteq y \Rightarrow y \doteq x$$
$$x \doteq y \text{ and } y \doteq z \Rightarrow x \doteq z$$
$$t_H \doteq \mathtt{L}$$
$$t_M \doteq \mathtt{L}$$
$$t_H = t_M \Rightarrow t_H \doteq t_M$$

Figure 24: Lump equality relation

# 3   Conclusion

Lazy and eager evaluation can be resolved transparently for common expressions at the boundaries between languages with unforced and forced values. This is more convenient than an explicit force operator that programmers must use manually by anticipating which expressions must be forced.

The approach this paper used for interoperation between three languages is not scalable. Values from each language can be directly converted to values of the other two languages and back. $n$ languages require $n*(n-1)$ conversion mappings between them. As the number of languages increases, the number of conversion mappings grows geometrically, which is unmaintainable. A better approach would be to make only two conversion mappings per language and chain them together to form a single path between any two languages, which would require only $n-1$ conversion mappings and grow linearly. Were this done for this model, the number of conversion mappings would be four instead of six.

# References

[1] Jacob Matthews and Robert Bruce Findler. Operational semantics for multi-language programs. *SIGPLAN Not.*, 42(1):3–10, 2007.