

Interoperation for Lazy and Eager Evaluation

William Faught

April 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 incompatible language features transparently at the boundaries between languages. To address part of this problem, we present a model of computation that resolves lazy and eager evaluation strategies.

Matthews and Findler presented [2] a model that enabled safe interoperation between statically and dynamically typed languages with parametric and ad-hoc polymorphism, respectively. We extend this model in various ways to demonstrate that it is insufficient to enable safe interoperation between eagerly and lazily evaluated languages, analyze the underlying problem, then introduce changes that resolve the fundamental interoperation incompatibility between eager and lazy languages.

Our model is a strict superset of that of Matthews and Findler. We introduce a third language to the model that is based on Haskell and is identical to the ML model

except for having lazy evaluation. To the Haskell and ML models we introduce fixed-point operations to restore Turing completeness, and to every language we introduce lists. Hereafter, we use the names of Haskell, ML, and Scheme to refer to their counterparts in our model.

2 Model of Computation

Lists are added to all three languages. Being lazy, Haskell does not evaluate function arguments or list construction operands. These three points constitute the set of incompatible strictness points between Haskell and ML and Haskell and Scheme. At these points in ML and Scheme, reducible expressions in Haskell boundaries must not be evaluated.

Since values are irreducible at all points, and since the expressions in Haskell boundaries are irreducible at some points and not others, Haskell boundaries are a new kind of value called an *unforced value*. Like thunks, unforced values can be forced to evaluate to values. The Haskell expressions in Haskell boundaries are forced to evaluate to Haskell values, then the Haskell values are converted to ML or Scheme values. ML and Scheme values are called *forced values* because any might be the result of forcing an unforced value. Forced values are a proper subset of unforced values because unforced values can only be at points where forced values can also be, but forced values can be at points where unforced values cannot. ML and Scheme reduction rules and evaluation contexts use unforced values at the incompatible strictness points to match against Haskell boundaries, and their evaluation contexts prevent evaluation within Haskell boundaries at those points.

Figure 1 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*

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

Proof. By structural induction. □

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

$$\begin{aligned}
& zeroes = \mathbf{fix} (\lambda x_H : \{N\}. \mathbf{cons} \bar{0} x_H) \\
& (\mathbf{hs} (\{N\} \rightarrow \{N\}) (\lambda x_S. x_S)) zeroes \quad \rightarrow \\
& (\lambda x'_H : \{N\}. \mathbf{hs} \{N\} ((\lambda x_S. x_S) (\mathbf{sh} \{N\} x'_H))) zeroes \quad \rightarrow \\
& \mathbf{hs} \{N\} ((\lambda x_S. x_S) (\mathbf{sh} \{N\} zeroes)) \quad \rightarrow \\
& \mathbf{hs} \{N\} (\mathbf{sh} \{N\} zeroes) \quad \rightarrow \\
& \mathbf{hs} \{N\} (\mathbf{sh} \{N\} (\mathbf{cons} \bar{0} zeroes)) \quad \rightarrow \\
& \mathbf{hs} \{N\} (\mathbf{cons} (\mathbf{sh} N \bar{0}) (\mathbf{sh} \{N\} zeroes)) \quad \rightarrow \\
& \mathbf{hs} \{N\} (\mathbf{cons} \bar{0} (\mathbf{sh} \{N\} zeroes)) \quad \rightarrow \\
& \mathbf{cons} (\mathbf{hs} N \bar{0}) (\mathbf{hs} \{N\} (\mathbf{sh} \{N\} zeroes))
\end{aligned}$$

Figure 1: Haskell argument and list conversions.

notion of the types of the outer language. 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 [2] 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 2-4; Haskell is presented in figures 5-9; ML is presented in figures 10-14; Scheme is presented in figures 15-19; the unbrand function is presented in figure 20; and the lump equality relation is presented in figure 21.

Symbol	Name
b	Brand
k	Conversion scheme
e	Expression
E	Forced evaluation context
w	Forced value
L	Lump
\doteq	Lump equality relation
\mathcal{E}	Meta evaluation context
\bar{n}	Natural number
\mathbb{N}	Natural number
\rightarrow	Reduction relation
t	Type
u	Type variable
Γ	Typing environment
\vdash	Typing relation
U	Unforced evaluation context
v	Unforced value
x	Variable

Figure 2: Symbol names

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

Figure 3: Syntax names

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

Figure 4: Syntax names

$$\begin{aligned}
e_H &= x_H \mid v_H \mid e_H e_H \mid e_H \langle t_H \rangle \mid \mathbf{fix} \, e_H \mid o \, e_H e_H \mid \mathbf{if0} \, e_H e_H e_H \mid f \, e_H \\
&\quad \mathbf{null?} \, e_H \mid \mathbf{wrong} \, t_H \, string \mid \mathbf{hm} \, t_H \, t_M \, e_M \mid \mathbf{hs} \, k_H \, e_S \\
v_H &= \lambda x_H : t_H . e_H \mid \Lambda u_H . e_H \mid \bar{n} \mid \mathbf{nil} \, t_H \mid \mathbf{cons} \, e_H e_H \mid \mathbf{hm} \, L \, t_M \, w_M \\
&\quad \mathbf{hs} \, L \, w_S \\
t_H &= L \mid N \mid u_H \mid \{t_H\} \mid t_H \rightarrow t_H \mid \forall u_H . t_H \\
k_H &= L \mid N \mid u_H \mid \{k_H\} \mid k_H \rightarrow k_H \mid \forall u_H . k_H \mid b \diamond t_H \\
o &= + \mid - \\
f &= \mathbf{hd} \mid \mathbf{tl} \\
E_H &= []_H \mid E_H e_H \mid E_H \langle t_H \rangle \mid \mathbf{fix} \, E_H \mid o \, E_H e_H \mid o \, v_H \, E_H \\
&\quad \mathbf{if0} \, E_H e_H e_H \mid f \, E_H \mid \mathbf{null?} \, E_H \mid \mathbf{hm} \, t_H \, t_M \, E_M \mid \mathbf{hs} \, k_H \, E_S
\end{aligned}$$

Figure 5: Haskell syntax and evaluation contexts

$$\begin{array}{c}
\overline{\vdash_H \mathbf{L}} \quad \overline{\vdash_H \mathbf{N}} \quad \overline{\Gamma, u_H \vdash_H u_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 \rightarrow t'_H} \quad \frac{\Gamma, u_H \vdash_H t_H}{\Gamma \vdash_H \forall u_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 \rightarrow t'_H} \quad \frac{\Gamma, u_H \vdash_H e_H : t_H}{\Gamma \vdash_H \Lambda u_H. e_H : \forall u_H. t_H} \quad \overline{\vdash_H \overline{n} : \mathbf{N}} \\
\frac{\Gamma \vdash_H t_H : \quad \Gamma \vdash_H e_H : t_H \quad \Gamma \vdash_H e'_H : \{t_H\}}{\Gamma \vdash_H \mathbf{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 \mathbf{cons} \ e_H \ e'_H : \{t_H\}} \quad \frac{\Gamma, x_H : t_H \vdash_H x_H : t_H}{\Gamma \vdash_H \mathbf{cons} \ e_H \ e'_H : \{t_H\}} \\
\frac{\Gamma \vdash_H e_H : t_H \rightarrow 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 \rightarrow t_H}{\Gamma \vdash_H \mathbf{fix} \ e_H : t_H} \\
\frac{\Gamma \vdash_H t_H \quad \Gamma \vdash_H e_H : \forall u_H. t'_H}{\Gamma \vdash_H e_H \langle t_H \rangle : t'_H[t_H/u_H]} \quad \frac{\Gamma \vdash_H e_H : \{t_H\}}{\Gamma \vdash_H \mathbf{hd} \ e_H : t_H} \quad \frac{\Gamma \vdash_H e_H : \{t_H\}}{\Gamma \vdash_H \mathbf{tl} \ e_H : \{t_H\}} \\
\frac{\Gamma \vdash_H e_H : \mathbf{N} \quad \Gamma \vdash_H e'_H : \mathbf{N}}{\Gamma \vdash_H o \ e_H \ e'_H : \mathbf{N}} \quad \frac{\Gamma \vdash_H e_H : \{t_H\}}{\Gamma \vdash_H \mathbf{null?} \ e_H : \mathbf{N}} \quad \frac{\Gamma \vdash_H [k_H] \quad \Gamma \vdash_S e_S : \mathbf{TST}}{\Gamma \vdash_H \mathbf{hs} \ k_H \ e_S : [k_H]} \\
\frac{\Gamma \vdash_H e_H : \mathbf{N} \quad \Gamma \vdash_H e'_H : t_H \quad \Gamma \vdash_H e''_H : t_H}{\Gamma \vdash_H \mathbf{if0} \ e_H \ e'_H \ e''_H : t_H} \quad \frac{\Gamma \vdash_H t_H}{\Gamma \vdash_H \mathbf{wrong} \ t_H \ \textit{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 \mathbf{hm} \ t_H \ t_M \ e_M : t_H}
\end{array}$$

Figure 6: Haskell typing rules

$$\begin{aligned}
& \mathcal{E}[(\lambda x_H : t_H.e_H) e'_H]_H \rightarrow \mathcal{E}[e_H[e'_H/x_H]] \\
& \mathcal{E}[(\Lambda u_H.e_H)(t_H)]_H \rightarrow \mathcal{E}[e_H[b \diamond t_H/u_H]] \\
& \mathcal{E}[\mathbf{fix} (\lambda x_H : t_H.e_H)]_H \rightarrow \mathcal{E}[e_H[\mathbf{fix} (\lambda x_H : t_H.e_H)/x_H]] \\
& \mathcal{E}[+ \bar{n} \bar{n'}]_H \rightarrow \mathcal{E}[\overline{n + n'}] \\
& \mathcal{E}[- \bar{n} \bar{n'}]_H \rightarrow \mathcal{E}[\overline{\max(n - n', 0)}] \\
& \mathcal{E}[\mathbf{if0} \bar{0} e_H e'_H]_H \rightarrow \mathcal{E}[e_H] \\
& \mathcal{E}[\mathbf{if0} \bar{n} e_H e'_H]_H \rightarrow \mathcal{E}[e'_H] \ (n \neq 0) \\
& \mathcal{E}[\mathbf{hd} (\mathbf{nil} t_H)]_H \rightarrow \mathcal{E}[\mathbf{wrong} t_H \text{ “Empty list”}] \\
& \mathcal{E}[\mathbf{tl} (\mathbf{nil} t_H)]_H \rightarrow \mathcal{E}[\mathbf{wrong} \{t_H\} \text{ “Empty list”}] \\
& \mathcal{E}[\mathbf{hd} (\mathbf{cons} e_H e'_H)]_H \rightarrow \mathcal{E}[e_H] \\
& \mathcal{E}[\mathbf{tl} (\mathbf{cons} e_H e'_H)]_H \rightarrow \mathcal{E}[e'_H] \\
& \mathcal{E}[\mathbf{null?} (\mathbf{nil} t_H)]_H \rightarrow \mathcal{E}[\bar{0}] \\
& \mathcal{E}[\mathbf{null?} (\mathbf{cons} e_H e'_H)]_H \rightarrow \mathcal{E}[\bar{1}] \\
& \mathcal{E}[\mathbf{wrong} t_H \text{ string}]_H \rightarrow \mathbf{Error: string}
\end{aligned}$$

Figure 7: Haskell operational semantics

$$\begin{aligned}
& \mathcal{E}[\mathbf{hm} \ t_H \ \mathbf{L} \ (\mathbf{mh} \ \mathbf{L} \ t'_H \ e_H)]_H \rightarrow \mathcal{E}[e_H] \quad (t_H = t'_H \text{ and } t_H \neq \mathbf{L}) \\
& \mathcal{E}[\mathbf{hm} \ t_H \ \mathbf{L} \ (\mathbf{mh} \ \mathbf{L} \ t'_H \ e_H)]_H \rightarrow \mathcal{E}[\mathbf{wrong} \ t_H \ \text{“Type mismatch”}] \\
& \quad (t_H \neq t'_H \text{ and } t_H \neq \mathbf{L}) \\
& \mathcal{E}[\mathbf{hm} \ t_H \ \mathbf{L} \ (\mathbf{ms} \ \mathbf{L} \ w_S)]_H \rightarrow \mathcal{E}[\mathbf{wrong} \ t_H \ \text{“Bad value”}] \quad (t_H \neq \mathbf{L}) \\
& \mathcal{E}[\mathbf{hm} \ \mathbf{N} \ \mathbf{N} \ \overline{n}]_H \rightarrow \mathcal{E}[\overline{n}] \\
& \mathcal{E}[\mathbf{hm} \ \{t_H\} \ \{t_M\} \ (\mathbf{nil} \ t'_M)]_H \rightarrow \mathcal{E}[\mathbf{nil} \ t_H] \\
& \mathcal{E}[\mathbf{hm} \ \{t_H\} \ \{t_M\} \ (\mathbf{cons} \ v_M \ v'_M)]_H \rightarrow \\
& \quad \mathcal{E}[\mathbf{cons} \ (\mathbf{hm} \ t_H \ t_M \ v_M) \ (\mathbf{hm} \ \{t_H\} \ \{t_M\} \ v'_M)] \\
& \mathcal{E}[\mathbf{hm} \ (t_H \rightarrow t'_H) \ (t_M \rightarrow t'_M) \ (\lambda x_M : t''_M . e_M)]_H \rightarrow \\
& \quad \mathcal{E}[\lambda x_H : t_H . \mathbf{hm} \ t'_H \ t'_M \ ((\lambda x_M : t''_M . e_M) \ (\mathbf{mh} \ t_M \ t_H \ x_H))] \\
& \mathcal{E}[\mathbf{hm} \ (\forall u_H . t_H) \ (\forall u_M . t_M) \ (\Lambda u'_M . e_M)]_H \rightarrow \mathcal{E}[\Lambda u_H . \mathbf{hm} \ t_H \ t_M [\mathbf{L}/u_M] \ e_M [\mathbf{L}/u'_M]]
\end{aligned}$$

Figure 8: Haskell-ML operational semantics

$$\begin{aligned}
& \mathcal{E}[\mathbf{hs} \ N \ \bar{n}]_H \rightarrow \mathcal{E}[\bar{n}] \\
& \mathcal{E}[\mathbf{hs} \ N \ w_S]_H \rightarrow \mathcal{E}[\mathbf{wrong} \ N \ \text{“Not a number”}] \ (w_S \neq \bar{n}) \\
& \mathcal{E}[\mathbf{hs} \ \{k_H\} \ \mathbf{nil}]_H \rightarrow \mathcal{E}[\mathbf{nil} \ [k_H]] \\
& \mathcal{E}[\mathbf{hs} \ \{k_H\} \ (\mathbf{cons} \ v_S \ v'_S)]_H \rightarrow \mathcal{E}[\mathbf{cons} \ (\mathbf{hs} \ k_H \ v_S) \ (\mathbf{hs} \ \{k_H\} \ v'_S)] \\
& \mathcal{E}[\mathbf{hs} \ \{k_H\} \ w_S]_H \rightarrow \mathcal{E}[\mathbf{wrong} \ [\{k_H\}] \ \text{“Not a list”}] \\
& \quad (w_S \neq \mathbf{nil} \text{ and } w_S \neq \mathbf{cons} \ v_S \ v'_S) \\
& \mathcal{E}[\mathbf{hs} \ (b \diamond t_H) \ (\mathbf{sh} \ (b \diamond t_H) \ e_H)]_H \rightarrow \mathcal{E}[e_H] \\
& \mathcal{E}[\mathbf{hs} \ (b \diamond t_H) \ w_S]_H \rightarrow \mathcal{E}[\mathbf{wrong} \ t_H \ \text{“Brand mismatch”}] \ (w_S \neq \mathbf{sh} \ (b \diamond t_H) \ e_H) \\
& \mathcal{E}[\mathbf{hs} \ (k_H \rightarrow k'_H) \ (\lambda x_S. e_S)]_H \rightarrow \mathcal{E}[\lambda x_H : [k_H]. \mathbf{hs} \ k'_H \ ((\lambda x_S. e_S) \ (\mathbf{sh} \ k_H \ x_H))] \\
& \mathcal{E}[\mathbf{hs} \ (k_H \rightarrow k'_H) \ w_S]_H \rightarrow \mathcal{E}[\mathbf{wrong} \ [k_H \rightarrow k'_H] \ \text{“Not a function”}] \\
& \quad (w_S \neq \lambda x_S. e_S) \\
& \mathcal{E}[\mathbf{hs} \ (\forall u_H. k_H) \ w_S]_H \rightarrow \mathcal{E}[\Lambda u_H. \mathbf{hs} \ k_H \ w_S]
\end{aligned}$$

Figure 9: Haskell-Scheme operational semantics

$$\begin{aligned}
e_M &= x_M \mid v_M \mid e_M e_M \mid e_M \langle t_M \rangle \mid \mathbf{fix} \ e_M \mid o \ e_M \ e_M \mid \mathbf{if0} \ e_M \ e_M \ e_M \\
&\quad \mathbf{cons} \ e_M \ e_M \mid f \ e_M \mid \mathbf{null?} \ e_M \mid \mathbf{wrong} \ t_M \ string \mid \mathbf{ms} \ k_M \ e_S \\
v_M &= w_M \mid \mathbf{mh} \ t_M \ t_H \ e_H \\
w_M &= \lambda x_M : t_M . e_M \mid \Lambda u_M . e_M \mid \bar{n} \mid \mathbf{nil} \ t_M \mid \mathbf{cons} \ v_M \ v_M \mid \mathbf{mh} \ L \ t_H \ e_H \\
&\quad \mathbf{ms} \ L \ w_S \\
t_M &= L \mid N \mid u_M \mid \{t_M\} \mid t_M \rightarrow t_M \mid \forall u_M . t_M \\
k_M &= L \mid N \mid u_M \mid \{k_M\} \mid k_M \rightarrow k_M \mid \forall u_M . k_M \mid b \diamond t_M \\
o &= + \mid - \\
f &= \mathbf{hd} \mid \mathbf{tl} \\
E_M &= U_M \mid \mathbf{mh} \ t_M \ t_H \ E_H \\
U_M &= []_M \mid E_M \ e_M \mid w_M \ U_M \mid E_M \langle t_M \rangle \mid \mathbf{fix} \ E_M \mid o \ E_M \ e_M \mid o \ w_M \ E_M \\
&\quad \mathbf{if0} \ E_M \ e_M \ e_M \mid \mathbf{cons} \ U_M \ e_M \mid \mathbf{cons} \ v_M \ U_M \mid f \ E_M \mid \mathbf{null?} \ E_M \\
&\quad \mathbf{ms} \ k_M \ E_S
\end{aligned}$$

Figure 10: ML syntax and evaluation contexts

$$\begin{array}{c}
\overline{\vdash_M \mathbf{L}} \quad \overline{\vdash_M \mathbf{N}} \quad \overline{\Gamma, u_M \vdash_M u_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 \rightarrow t'_M} \quad \frac{\Gamma, u_M \vdash_M t_M}{\Gamma \vdash_M \forall u_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 \rightarrow t'_M} \quad \frac{\Gamma, u_M \vdash_M e_M : t_M}{\Gamma \vdash_M \Lambda u_M. e_M : \forall u_M. t_M} \quad \overline{\vdash_M \bar{n} : \mathbf{N}} \\
\\
\frac{\Gamma \vdash_M t_M}{\Gamma \vdash_M \mathbf{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 \mathbf{cons} \ e_M \ e'_M : \{t_M\}} \quad \frac{}{\Gamma, x_M : t_M \vdash_M x_M : t_M} \\
\\
\frac{\Gamma \vdash_M e_M : t_M \rightarrow 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 \rightarrow t_M}{\Gamma \vdash_M \mathbf{fix} \ e_M : t_M} \\
\\
\frac{\Gamma \vdash_M t_M \quad \Gamma \vdash_M e_M : \forall u_M. t'_M}{\Gamma \vdash_M e_M \langle t_M \rangle : t'_M[t_M/u_M]} \quad \frac{\Gamma \vdash_M e_M : \{t_M\}}{\Gamma \vdash_M \mathbf{hd} \ e_M : t_M} \quad \frac{\Gamma \vdash_M e_M : \{t_M\}}{\Gamma \vdash_M \mathbf{tl} \ e_M : \{t_M\}} \\
\\
\frac{\Gamma \vdash_M e_M : \mathbf{N} \quad \Gamma \vdash_M e'_M : \mathbf{N}}{\Gamma \vdash_M o \ e_M \ e'_M : \mathbf{N}} \quad \frac{\Gamma \vdash_M e_M : \{t_M\}}{\Gamma \vdash_M \mathbf{null?} \ e_M : \mathbf{N}} \quad \frac{\Gamma \vdash_M \lfloor k_M \rfloor \quad \Gamma \vdash_S e_S : \mathbf{TST}}{\Gamma \vdash_M \mathbf{ms} \ k_M \ e_S : \lfloor k_M \rfloor} \\
\\
\frac{\Gamma \vdash_M e_M : \mathbf{N} \quad \Gamma \vdash_M e'_M : t_M \quad \Gamma \vdash_M e''_M : t_M}{\Gamma \vdash_M \mathbf{if0} \ e_M \ e'_M \ e''_M : t_M} \quad \frac{\Gamma \vdash_M t_M}{\Gamma \vdash_M \mathbf{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 \mathbf{mh} \ t_M \ t_H \ e_H : t_M}
\end{array}$$

Figure 11: ML typing rules

$$\begin{aligned}
& \mathcal{E}[(\lambda x_M : t_M.e_M) v_M]_M \rightarrow \mathcal{E}[e_M[v_M/x_M]] \\
& \mathcal{E}[(\Lambda u_M.e_M)\langle t_M \rangle]_M \rightarrow \mathcal{E}[e_M[b \diamond t_M/u_M]] \\
& \mathcal{E}[\mathbf{fix} (\lambda x_M : t_M.e_M)]_M \rightarrow \mathcal{E}[e_M[\mathbf{fix} (\lambda x_M : t_M.e_M)/x_M]] \\
& \mathcal{E}[+ \bar{n} \bar{n}']_M \rightarrow \mathcal{E}[\overline{n + n'}] \\
& \mathcal{E}[- \bar{n} \bar{n}']_M \rightarrow \mathcal{E}[\overline{\max(n - n', 0)}] \\
& \mathcal{E}[\mathbf{if0} \bar{0} e_M e'_M]_M \rightarrow \mathcal{E}[e_M] \\
& \mathcal{E}[\mathbf{if0} \bar{n} e_M e'_M]_M \rightarrow \mathcal{E}[e'_M] \quad (n \neq 0) \\
& \mathcal{E}[\mathbf{hd} (\mathbf{nil} t_M)]_M \rightarrow \mathcal{E}[\mathbf{wrong} t_M \text{ “Empty list”}] \\
& \mathcal{E}[\mathbf{tl} (\mathbf{nil} t_M)]_M \rightarrow \mathcal{E}[\mathbf{wrong} \{t_M\} \text{ “Empty list”}] \\
& \mathcal{E}[\mathbf{hd} (\mathbf{cons} v_M v'_M)]_M \rightarrow \mathcal{E}[v_M] \\
& \mathcal{E}[\mathbf{tl} (\mathbf{cons} v_M v'_M)]_M \rightarrow \mathcal{E}[v'_M] \\
& \mathcal{E}[\mathbf{null?} (\mathbf{nil} t_M)]_M \rightarrow \mathcal{E}[\bar{0}] \\
& \mathcal{E}[\mathbf{null?} (\mathbf{cons} v_M v'_M)]_M \rightarrow \mathcal{E}[\bar{1}] \\
& \mathcal{E}[\mathbf{wrong} t_M \text{ string}]_H \rightarrow \mathbf{Error: string}
\end{aligned}$$

Figure 12: ML operational semantics

$$\begin{aligned}
\mathcal{E}[\mathbf{mh} \ t_M \ \mathbf{L} \ (\mathbf{hm} \ \mathbf{L} \ t'_M \ w_M)]_M &\rightarrow \mathcal{E}[w_M] \ (t_M = t'_M \text{ and } t_M \neq \mathbf{L}) \\
\mathcal{E}[\mathbf{mh} \ t_M \ \mathbf{L} \ (\mathbf{hm} \ \mathbf{L} \ t'_M \ w_M)]_M &\rightarrow \mathcal{E}[\mathbf{wrong} \ t_M \ \text{“Type mismatch”}] \ (t_M \neq t'_M \text{ and } t_M \neq \mathbf{L}) \\
\mathcal{E}[\mathbf{mh} \ t_M \ \mathbf{L} \ (\mathbf{hs} \ \mathbf{L} \ w_S)]_H &\rightarrow \mathcal{E}[\mathbf{wrong} \ t_M \ \text{“Bad value”}] \ (t_M \neq \mathbf{L}) \\
\mathcal{E}[\mathbf{mh} \ \mathbf{N} \ \mathbf{N} \ \bar{n}]_M &\rightarrow \mathcal{E}[\bar{n}] \\
\mathcal{E}[\mathbf{mh} \ \{t_M\} \ \{t_H\} \ (\mathbf{nil} \ t'_H)]_M &\rightarrow \mathcal{E}[\mathbf{nil} \ t_M] \\
\mathcal{E}[\mathbf{mh} \ \{t_M\} \ \{t_H\} \ (\mathbf{cons} \ e_H \ e'_H)]_M &\rightarrow \mathcal{E}[\mathbf{cons} \ (\mathbf{mh} \ t_M \ t_H \ e_H) \ (\mathbf{mh} \ \{t_M\} \ \{t_H\} \ e'_H)] \\
\mathcal{E}[\mathbf{mh} \ (t_M \rightarrow t'_M) \ (t_H \rightarrow t'_H) \ (\lambda x_H : t''_H . e_H)]_M &\rightarrow \\
&\mathcal{E}[\lambda x_M : t_M . \mathbf{mh} \ t'_M \ t'_H \ ((\lambda x_H : t''_H . e_H) \ (\mathbf{hm} \ t_H \ t_M \ x_M))] \\
\mathcal{E}[\mathbf{mh} \ (\forall u_M . t_M) \ (\forall u_H . t_H) \ (\Lambda u'_H . e_H)]_M &\rightarrow \mathcal{E}[\Lambda u_M . \mathbf{mh} \ t_M \ t_H [\mathbf{L}/u_H] \ e_H [\mathbf{L}/u'_H]]
\end{aligned}$$

Figure 13: ML-Haskell operational semantics

$$\begin{aligned}
& \mathcal{E}[\mathbf{ms} \ \mathbf{N} \ \bar{n}]_M \rightarrow \mathcal{E}[\bar{n}] \\
& \mathcal{E}[\mathbf{ms} \ \mathbf{N} \ w_S]_M \rightarrow \mathcal{E}[\mathbf{wrong} \ \mathbf{N} \ \text{“Not a number”}] \ (w_S \neq \bar{n}) \\
& \mathcal{E}[\mathbf{ms} \ \{k_M\} \ \mathbf{nil}]_M \rightarrow \mathcal{E}[\mathbf{nil} \ \lfloor k_M \rfloor] \\
& \mathcal{E}[\mathbf{ms} \ \{k_M\} \ (\mathbf{cons} \ v_S \ v'_S)]_M \rightarrow \mathcal{E}[\mathbf{cons} \ (\mathbf{ms} \ k_M \ v_S) \ (\mathbf{ms} \ \{k_M\} \ v'_S)] \\
& \mathcal{E}[\mathbf{ms} \ \{k_M\} \ w_S]_M \rightarrow \mathcal{E}[\mathbf{wrong} \ \lfloor \{k_M\} \rfloor \ \text{“Not a list”}] \\
& \quad (w_S \neq \mathbf{nil} \text{ and } w_S \neq \mathbf{cons} \ v_S \ v'_S) \\
& \mathcal{E}[\mathbf{ms} \ (b \diamond t_M) \ (\mathbf{sm} \ (b \diamond t_M) \ v_M)]_M \rightarrow \mathcal{E}[v_M] \\
& \mathcal{E}[\mathbf{ms} \ (b \diamond t_M) \ w_S]_M \rightarrow \mathcal{E}[\mathbf{wrong} \ \lfloor b \diamond t_M \rfloor \ \text{“Brand mismatch”}] \\
& \quad (w_S \neq \mathbf{sm} \ (b \diamond t_M) \ e_M) \\
& \mathcal{E}[\mathbf{ms} \ (k_M \rightarrow k'_M) \ (\lambda x_S. e_S)]_M \rightarrow \\
& \quad \mathcal{E}[\lambda x_M : \lfloor k_M \rfloor. \mathbf{ms} \ k'_M \ ((\lambda x_S. e_S) \ (\mathbf{sm} \ k_M \ x_M))] \\
& \mathcal{E}[\mathbf{ms} \ (k_M \rightarrow k'_M) \ w_S]_M \rightarrow \mathcal{E}[\mathbf{wrong} \ \lfloor k_M \rightarrow k'_M \rfloor \ \text{“Not a function”}] \\
& \quad (w_S \neq \lambda x_S. e_S) \\
& \mathcal{E}[\mathbf{ms} \ (\forall u_M. k_M) \ w_S]_M \rightarrow \mathcal{E}[\Lambda u_M. \mathbf{ms} \ k_M \ w_S]
\end{aligned}$$

Figure 14: ML-Scheme operational semantics

$$\begin{aligned}
e_S &= x_S \mid v_S \mid e_S e_S \mid o e_S e_S \mid p e_S \mid \text{if0 } e_S e_S e_S \mid \text{cons } e_S e_S \mid f e_S \\
&\quad \text{wrong } string \mid \text{sm } k_M e_M \\
v_S &= w_S \mid \text{sh } k_H e_H \\
w_S &= \lambda x_S. e_S \mid \bar{n} \mid \text{nil} \mid \text{cons } v_S v_S \mid \text{sh } (b \diamond t_H) e_H \mid \text{sm } (b \diamond t_M) w_M \\
o &= + \mid - \\
f &= \text{hd} \mid \text{tl} \\
p &= \text{fun?} \mid \text{list?} \mid \text{null?} \mid \text{num?} \\
E_S &= U_S \mid \text{sh } k_H E_H \\
U_S &= []_S \mid E_S e_S \mid w_S U_S \mid o E_S e_S \mid o w_S E_S \mid p E_S \mid \text{if0 } E_S e_S e_S \\
&\quad \text{cons } U_S e_S \mid \text{cons } v_S U_S \mid f E_S \mid \text{sm } k_M E_M
\end{aligned}$$

Figure 15: Scheme syntax and evaluation contexts

$$\begin{array}{c}
\overline{\vdash_S \text{TST}} \\
\\
\frac{\Gamma, x_S : \text{TST} \vdash_S e_S : \text{TST}}{\Gamma \vdash_S \lambda x_S. e_S : \text{TST}} \quad \overline{\vdash_S \bar{n} : \text{TST}} \quad \overline{\vdash_S \text{nil} : \text{TST}} \\
\frac{\Gamma \vdash_S e_S : \text{TST} \quad \Gamma \vdash_S e'_S : \text{TST}}{\Gamma \vdash_S \text{cons } e_S e'_S : \text{TST}} \quad \overline{\Gamma, x_S : \text{TST} \vdash_S x_S : \text{TST}} \\
\frac{\Gamma \vdash_S e_S : \text{TST} \quad \Gamma \vdash_S e'_S : \text{TST}}{\Gamma \vdash_H e_S e'_S : \text{TST}} \quad \frac{\Gamma \vdash_S e_S : \text{TST}}{\Gamma \vdash_S f e_S : \text{TST}} \\
\frac{\Gamma \vdash_S e_S : \text{TST} \quad \Gamma \vdash_S e'_S : \text{TST}}{\Gamma \vdash_S o e_S e'_S : \text{TST}} \quad \frac{\Gamma \vdash_S e_S : \text{TST}}{\Gamma \vdash_S p e_S : \text{TST}} \\
\frac{\Gamma \vdash_S e_S : \text{TST} \quad \Gamma \vdash_S e'_S : \text{TST} \quad \Gamma \vdash_S e''_S : \text{TST}}{\Gamma \vdash_S \text{if0 } e_S e'_S e''_S : \text{TST}} \quad \overline{\vdash_S \text{wrong string} : \text{TST}} \\
\frac{\Gamma \vdash_H [k_H] \quad \Gamma \vdash_H e_H : t_H \quad [k_H] = t_H}{\Gamma \vdash_S \text{sh } k_H e_H : \text{TST}} \quad \frac{\Gamma \vdash_M [k_M] \quad \Gamma \vdash_M e_M : t_M \quad [k_M] = t_M}{\Gamma \vdash_S \text{sm } k_M e_M : \text{TST}}
\end{array}$$

Figure 16: Scheme typing rules

$$\begin{aligned}
& \mathcal{E}[(\lambda x_S. e_S) v_S]_S \rightarrow \mathcal{E}[e_S[v_S/x_S]] \\
& \mathcal{E}[w_S v_S]_S \rightarrow \mathcal{E}[\mathbf{wrong} \text{ “Not a function”}] \ (w_S \neq \lambda x_S. e_S) \\
& \mathcal{E}[+ \bar{n} \bar{n}']_S \rightarrow \mathcal{E}[\overline{n + n'}] \\
& \mathcal{E}[- \bar{n} \bar{n}']_S \rightarrow \mathcal{E}[\overline{\max(n - n', 0)}] \\
& \mathcal{E}[o w_S w'_S]_S \rightarrow \mathcal{E}[\mathbf{wrong} \text{ “Not a number”}] \ (w_S \neq \bar{n} \text{ or } w'_S \neq \bar{n}) \\
& \mathcal{E}[\mathbf{if0} \ \bar{0} \ e_S \ e'_S]_S \rightarrow \mathcal{E}[e_S] \\
& \mathcal{E}[\mathbf{if0} \ \bar{n} \ e_S \ e'_S]_S \rightarrow \mathcal{E}[e'_S] \ (n \neq 0) \\
& \mathcal{E}[\mathbf{if0} \ w_S \ e_S \ e'_S]_S \rightarrow \mathcal{E}[\mathbf{wrong} \text{ “Not a number”}] \ (w_S \neq \bar{n}) \\
& \mathcal{E}[f \ \mathbf{nil}]_S \rightarrow \mathcal{E}[\mathbf{wrong} \text{ “Empty list”}] \\
& \mathcal{E}[\mathbf{hd} \ (\mathbf{cons} \ v_S \ v'_S)]_S \rightarrow \mathcal{E}[v_S] \\
& \mathcal{E}[\mathbf{tl} \ (\mathbf{cons} \ v_S \ v'_S)]_S \rightarrow \mathcal{E}[v'_S] \\
& \mathcal{E}[f \ w_S]_S \rightarrow \mathcal{E}[\mathbf{wrong} \text{ “Not a list”}] \ (w_S \neq \mathbf{nil} \text{ and } w_S \neq \mathbf{cons} \ v_S \ v'_S) \\
& \mathcal{E}[\mathbf{fun?} \ (\lambda x_S. e_S)]_S \rightarrow \mathcal{E}[\bar{0}] \\
& \mathcal{E}[\mathbf{fun?} \ w_S]_S \rightarrow \mathcal{E}[\bar{1}] \ (w_S \neq \lambda x_S. e_S) \\
& \mathcal{E}[\mathbf{list?} \ \mathbf{nil}]_S \rightarrow \mathcal{E}[\bar{0}] \\
& \mathcal{E}[\mathbf{list?} \ (\mathbf{cons} \ v_S \ v'_S)]_S \rightarrow \mathcal{E}[\bar{0}] \\
& \mathcal{E}[\mathbf{list?} \ w_S]_S \rightarrow \mathcal{E}[\bar{1}] \ (w_S \neq \mathbf{nil} \text{ and } w_S \neq \mathbf{cons} \ v_S \ v'_S) \\
& \mathcal{E}[\mathbf{null?} \ \mathbf{nil}]_S \rightarrow \mathcal{E}[\bar{0}] \\
& \mathcal{E}[\mathbf{null?} \ w_S]_S \rightarrow \mathcal{E}[\bar{1}] \ (w_S \neq \mathbf{nil}) \\
& \mathcal{E}[\mathbf{num?} \ \bar{n}]_S \rightarrow \mathcal{E}[\bar{0}] \\
& \mathcal{E}[\mathbf{num?} \ w_S]_S \rightarrow \mathcal{E}[\bar{1}] \ (w_S \neq \bar{n}) \\
& \mathcal{E}[\mathbf{wrong} \ string]_S \rightarrow \mathbf{Error:} \ string
\end{aligned}$$

Figure 17: Scheme operational semantics

$$\begin{aligned}
\mathcal{E}[\mathbf{sh}\ L\ (\mathbf{hm}\ L\ k_M\ w_M)]_S &\rightarrow \mathcal{E}[\mathbf{wrong}\ \text{“Bad value”}] \\
\mathcal{E}[\mathbf{sh}\ L\ (\mathbf{hs}\ L\ w_S)]_S &\rightarrow \mathcal{E}[w_S] \\
\mathcal{E}[\mathbf{sh}\ N\ \bar{n}]_S &\rightarrow \mathcal{E}[\bar{n}] \\
\mathcal{E}[\mathbf{sh}\ \{k_H\}\ (\mathbf{nil}\ t_H)]_S &\rightarrow \mathcal{E}[\mathbf{nil}] \\
\mathcal{E}[\mathbf{sh}\ \{k_H\}\ (\mathbf{cons}\ e_H\ e'_H)]_S &\rightarrow \mathcal{E}[\mathbf{cons}\ (\mathbf{sh}\ k_H\ e_H)\ (\mathbf{sh}\ \{k_H\}\ e'_H)] \\
\mathcal{E}[\mathbf{sh}\ (k_H \rightarrow k'_H)\ (\lambda x_H : t_H.e_H)]_S &\rightarrow \\
&\quad \mathcal{E}[\lambda x_S.\mathbf{sh}\ k'_H\ ((\lambda x_H : t_H.e_H)\ (\mathbf{hs}\ k_H\ x_S))] \\
\mathcal{E}[\mathbf{sh}\ (\forall u_H.k_H)\ (\Lambda u'_H.e_H)]_S &\rightarrow \mathcal{E}[\mathbf{sh}\ k_H[L/u_H]\ e_H[L/u'_H]]
\end{aligned}$$

Figure 18: Scheme-Haskell operational semantics

$$\begin{aligned}
& \mathcal{E}[\mathbf{sm} \ L \ (\mathbf{mh} \ L \ k_H \ e_H)]_S \rightarrow \mathcal{E}[\mathbf{wrong} \ \text{“Bad value”}] \\
& \mathcal{E}[\mathbf{sm} \ L \ (\mathbf{ms} \ L \ w_S)]_S \rightarrow \mathcal{E}[w_S] \\
& \mathcal{E}[\mathbf{sm} \ N \ \bar{n}]_S \rightarrow \mathcal{E}[\bar{n}] \\
& \mathcal{E}[\mathbf{sm} \ \{k_M\} \ (\mathbf{nil} \ t_M)]_S \rightarrow \mathcal{E}[\mathbf{nil}] \\
& \mathcal{E}[\mathbf{sm} \ \{k_M\} \ (\mathbf{cons} \ v_M \ v'_M)]_S \rightarrow \mathcal{E}[\mathbf{cons} \ (\mathbf{sm} \ k_M \ v_M) \ (\mathbf{sm} \ \{k_M\} \ v'_M)] \\
& \mathcal{E}[\mathbf{sm} \ (k_M \rightarrow k'_M) \ (\lambda x_M : t_M.e_M)]_S \rightarrow \\
& \quad \mathcal{E}[\lambda x_S. \mathbf{sm} \ k'_M \ ((\lambda x_M : t_M.e_M) \ (\mathbf{ms} \ k_M \ x_S))] \\
& \mathcal{E}[\mathbf{sm} \ (\forall u_M. k_M) \ (\Lambda u'_M. e_M)]_S \rightarrow \mathcal{E}[\mathbf{sm} \ k_M[L/u_M] \ e_M[L/u'_M]]
\end{aligned}$$

Figure 19: Scheme-ML operational semantics

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

Figure 20: Unbrand function

$$\begin{aligned}
& x \dot{=} x \\
& x \dot{=} y \Rightarrow y \dot{=} x \\
& x \dot{=} y \text{ and } y \dot{=} z \Rightarrow x \dot{=} z \\
& t_H \dot{=} L \\
& t_M \dot{=} L \\
& t_H = t_M \Rightarrow t_H \dot{=} t_M
\end{aligned}$$

Figure 21: 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] David L. Kinghorn. Preserving parametricity while sharing higher-order, polymorphic functions between scheme and ml. Master's thesis, California Polytechnic State University, San Luis Obispo, June 2007.
- [2] Jacob Matthews and Robert Bruce Findler. Operational semantics for multi-language programs. *SIGPLAN Not.*, 42(1):3–10, 2007.