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.

$$
\begin{array}{lll}
(\texttt{hs}\ (\texttt{N} \to \texttt{N})\ (\lambda x_S.x_S))\ \Omega & \to & (\text{by } F_H\ e_H) \\
(\lambda x_H : \texttt{N}.\texttt{hs}\ \texttt{N}\ ((\lambda x_S.x_S)\ (\texttt{sh}\ \texttt{N}\ x_H)))\ \Omega & \to & (\text{by } F_H) \\
\texttt{hs}\ \texttt{N}\ ((\lambda x_S.x_S)\ (\texttt{sh}\ \texttt{N}\ \Omega)) & \not\to & (\text{by } v_S\ E_S)
\end{array}
$$

Figure 1: Scheme forces the conversion of the argument.

$$
\begin{array}{ll}
(\texttt{hs}\ (\texttt{N} \to \texttt{N})\ (\lambda x_S.x_S))\ \Omega & \to \\
(\lambda x_H : \texttt{N}.\texttt{hs}\ \texttt{N}\ ((\lambda x_S.x_S)\ (\texttt{sh}\ \texttt{N}\ x_H)))\ \Omega & \to \\
\texttt{hs}\ \texttt{N}\ ((\lambda x_S.x_S)\ (\texttt{sh}\ \texttt{N}\ \Omega)) & \to \\
\texttt{hs}\ \texttt{N}\ (\texttt{sh}\ \texttt{N}\ \Omega) & \to \\
\Omega & \not\to
\end{array}
$$

Figure 2: Scheme does not force the conversion of the argument.

2 Model of Computation

The model is a strict superset of that of Matthews and Findler [2]. A third language is introduced to the model that is based on Haskell and is identical to the ML model except it is lazy. To the Haskell and ML models, fixed-point operations are introduced to restore Turing completeness, and lists are introduced to every language. Hereafter, we use the names of Haskell, ML, and Scheme to refer to their counterparts in the model.

Being lazy, Haskell does not evaluate function arguments or list construction operands. These three contexts constitute the set of incompatible strictness points between Haskell and ML, and Haskell and Scheme. In both cases, the eager language forces the evaluation of more expressions than the lazy one, hence the lazy one is less restrictive of the types of well-behaved expressions. Since the conversion of values between languages mirrors the original structure of the values, the eager languages cannot evaluate imported expressions in these contexts because they could in effect change the order of evaluation for those imported expressions from the perspective of the lazy language. In these contexts in ML and Scheme, reducible expressions in Haskell boundaries must not be evaluated.

$$
\begin{array}{ll}
\texttt{sh}\ \{\texttt{N}\}\ (\texttt{cons}\ \Omega_{\texttt{N}}\ \Omega_{\{\texttt{N}\}}) & \to \\
\texttt{cons}\ (\texttt{sh}\ \texttt{N}\ \Omega_{\texttt{N}})\ (\texttt{sh}\ \{\texttt{N}\}\ \Omega_{\{\texttt{N}\}}) & \not\to
\end{array}
$$

Figure 3: Scheme forces the conversion of a list construction operand.

$$
\begin{array}{ll}
\multicolumn{2}{c}{\mathit{zeroes} \equiv \texttt{fix}\ (\lambda x_H : \{\texttt{N}\}.\texttt{cons}\ \overline{0}\ x_H)} \\
(\texttt{hs}\ (\{\texttt{N}\} \rightarrow \{\texttt{N}\})\ (\lambda x_S.x_S))\ \mathit{zeroes} & \rightarrow \\
(\lambda x'_H : \{\texttt{N}\}.\texttt{hs}\ \{\texttt{N}\}\ ((\lambda x_S.x_S)\ (\texttt{sh}\ \{\texttt{N}\}\ x'_H)))\ \mathit{zeroes} & \rightarrow \\
\texttt{hs}\ \{\texttt{N}\}\ ((\lambda x_S.x_S)\ (\texttt{sh}\ \{\texttt{N}\}\ \mathit{zeroes})) & \rightarrow \\
\texttt{hs}\ \{\texttt{N}\}\ ((\lambda x_S.x_S)\ (\texttt{sh}\ \{\texttt{N}\}\ ((\lambda x_H : \{\texttt{N}\}.\texttt{cons}\ \overline{0}\ x_H)\ \mathit{zeroes}))) & \rightarrow \\
\texttt{hs}\ \{\texttt{N}\}\ ((\lambda x_S.x_S)\ (\texttt{sh}\ \{\texttt{N}\}\ (\texttt{cons}\ \overline{0}\ \mathit{zeroes}))) & \rightarrow \\
\texttt{hs}\ \{\texttt{N}\}\ ((\lambda x_S.x_S)\ (\texttt{cons}\ (\texttt{sh}\ \texttt{N}\ \overline{0})\ (\texttt{sh}\ \{\texttt{N}\}\ \mathit{zeroes}))) & \rightarrow \\
\texttt{hs}\ \{\texttt{N}\}\ ((\lambda x_S.x_S)\ (\texttt{cons}\ \overline{0}\ (\texttt{sh}\ \{\texttt{N}\}\ \mathit{zeroes}))) & \not\rightarrow
\end{array}
$$

Figure 4: Scheme does not force the conversion of list construction operands.

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

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

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 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 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
Γ	Typing environment
$\vdash$	Typing relation
U	Unforced evaluation context
u	Unforced value
x	Variable

Figure 5: Symbol names

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

Figure 6: Syntax names

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

Figure 7: Syntax names

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

Figure 8: Haskell syntax and evaluation contexts

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

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

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

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

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

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

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

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

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

Figure 9: Haskell typing rules

$$\mathscr{E}[(\lambda x_H : t_H.e_H)\ e'_H]_H \to \mathscr{E}[e_H[e'_H/x_H]]$$
$$\mathscr{E}[(\Lambda y_H.e_H)\langle t_H\rangle]_H \to \mathscr{E}[e_H[b \diamond t_H/y_H]]$$
$$\mathscr{E}[\texttt{fix}\ (\lambda x_H : t_H.e_H)]_H \to \mathscr{E}[e_H[\texttt{fix}\ (\lambda x_H : t_H.e_H)/x_H]]$$
$$\mathscr{E}[+\ \overline{n}\ \overline{n}']_H \to \mathscr{E}[\overline{n + n'}]$$
$$\mathscr{E}[-\ \overline{n}\ \overline{n}']_H \to \mathscr{E}[\overline{max(n - n', 0)}]$$
$$\mathscr{E}[\texttt{if0}\ \overline{0}\ e_H\ e'_H]_H \to \mathscr{E}[e_H]$$
$$\mathscr{E}[\texttt{if0}\ \overline{n}\ e_H\ e'_H]_H \to \mathscr{E}[e'_H]\ (n \neq 0)$$
$$\mathscr{E}[\texttt{hd}\ (\texttt{nil}\ t_H)]_H \to \mathscr{E}[\texttt{wrong}\ t_H\ \text{“Empty list”}]$$
$$\mathscr{E}[\texttt{tl}\ (\texttt{nil}\ t_H)]_H \to \mathscr{E}[\texttt{wrong}\ \{t_H\}\ \text{“Empty list”}]$$
$$\mathscr{E}[\texttt{hd}\ (\texttt{cons}\ e_H\ e'_H)]_H \to \mathscr{E}[e_H]$$
$$\mathscr{E}[\texttt{tl}\ (\texttt{cons}\ e_H\ e'_H)]_H \to \mathscr{E}[e'_H]$$
$$\mathscr{E}[\texttt{null?}\ (\texttt{nil}\ t_H)]_H \to \mathscr{E}[\overline{0}]$$
$$\mathscr{E}[\texttt{null?}\ (\texttt{cons}\ e_H\ e'_H)]_H \to \mathscr{E}[\overline{1}]$$
$$\mathscr{E}[\texttt{wrong}\ t_H\ \mathit{string}]_H \to \textbf{Error:}\ \mathit{string}$$

Figure 10: Haskell operational semantics

$\mathscr{E}[\texttt{hm}\ t_H\ \texttt{L}\ (\texttt{mh}\ \texttt{L}\ t'_H\ e_H)]_H \to \mathscr{E}[e_H] \quad (t_H = t'_H \text{ and } t_H \neq \texttt{L})$
$\mathscr{E}[\texttt{hm}\ t_H\ \texttt{L}\ (\texttt{mh}\ \texttt{L}\ t'_H\ e_H)]_H \to \mathscr{E}[\texttt{wrong}\ t_H\ \text{“Type mismatch”}]$
$\quad (t_H \neq t'_H \text{ and } t_H \neq \texttt{L})$
$\mathscr{E}[\texttt{hm}\ t_H\ \texttt{L}\ (\texttt{ms}\ \texttt{L}\ f_S)]_H \to \mathscr{E}[\texttt{wrong}\ t_H\ \text{“Bad value”}] \quad (t_H \neq \texttt{L})$
$\mathscr{E}[\texttt{hm}\ \texttt{N}\ \texttt{N}\ \overline{n}]_H \to \mathscr{E}[\overline{n}]$
$\mathscr{E}[\texttt{hm}\ \{t_H\}\ \{t_M\}\ (\texttt{nil}\ t'_M)]_H \to \mathscr{E}[\texttt{nil}\ t_H]$
$\mathscr{E}[\texttt{hm}\ \{t_H\}\ \{t_M\}\ (\texttt{cons}\ u_M\ u'_M)]_H \to$
$\quad \mathscr{E}[\texttt{cons}\ (\texttt{hm}\ t_H\ t_M\ u_M)\ (\texttt{hm}\ \{t_H\}\ \{t_M\}\ u'_M)]$
$\mathscr{E}[\texttt{hm}\ (t_H \to t'_H)\ (t_M \to t'_M)\ (\lambda x_M : t''_M.e_M)]_H \to$
$\quad \mathscr{E}[\lambda x_H : t_H.\texttt{hm}\ t'_H\ t'_M\ ((\lambda x_M : t''_M.e_M)\ (\texttt{mh}\ t_M\ t_H\ x_H))]$
$\mathscr{E}[\texttt{hm}\ (\forall y_H.t_H)\ (\forall y_M.t_M)\ (\Lambda y'_M.e_M)]_H \to \mathscr{E}[\Lambda y_H.\texttt{hm}\ t_H\ t_M[\texttt{L}/y_M]\ e_M[\texttt{L}/y'_M]]$

Figure 11: Haskell-ML operational semantics

$$\mathscr{E}[\texttt{hs N}\ \overline{n}]_H \rightarrow \mathscr{E}[\overline{n}]$$
$$\mathscr{E}[\texttt{hs N}\ f_S]_H \rightarrow \mathscr{E}[\texttt{wrong N}\ \text{“Not a number”}]\ (f_S \neq \overline{n})$$
$$\mathscr{E}[\texttt{hs}\ \{k_H\}\ \texttt{nil}]_H \rightarrow \mathscr{E}[\texttt{nil}\ \lfloor k_H \rfloor]$$
$$\mathscr{E}[\texttt{hs}\ \{k_H\}\ (\texttt{cons}\ u_S\ u'_S)]_H \rightarrow \mathscr{E}[\texttt{cons}\ (\texttt{hs}\ k_H\ u_S)\ (\texttt{hs}\ \{k_H\}\ u'_S)]$$
$$\mathscr{E}[\texttt{hs}\ \{k_H\}\ f_S]_H \rightarrow \mathscr{E}[\texttt{wrong}\ \lfloor \{k_H\} \rfloor\ \text{“Not a list”}]$$
$$(f_S \neq \texttt{nil}\ \text{and}\ f_S \neq \texttt{cons}\ u_S\ u'_S)$$
$$\mathscr{E}[\texttt{hs}\ (b \diamond t_H)\ (\texttt{sh}\ (b \diamond t_H)\ e_H)]_H \rightarrow \mathscr{E}[e_H]$$
$$\mathscr{E}[\texttt{hs}\ (b \diamond t_H)\ f_S]_H \rightarrow \mathscr{E}[\texttt{wrong}\ t_H\ \text{“Brand mismatch”}]\ (f_S \neq \texttt{sh}\ (b \diamond t_H)\ e_H)$$
$$\mathscr{E}[\texttt{hs}\ (k_H \rightarrow k'_H)\ (\lambda x_S.e_S)]_H \rightarrow \mathscr{E}[\lambda x_H : \lfloor k_H \rfloor.\texttt{hs}\ k'_H\ ((\lambda x_S.e_S)\ (\texttt{sh}\ k_H\ x_H))]$$
$$\mathscr{E}[\texttt{hs}\ (k_H \rightarrow k'_H)\ f_S]_H \rightarrow \mathscr{E}[\texttt{wrong}\ \lfloor k_H \rightarrow k'_H \rfloor\ \text{“Not a function”}]$$
$$(f_S \neq \lambda x_S.e_S)$$
$$\mathscr{E}[\texttt{hs}\ (\forall u_H.k_H)\ f_S]_H \rightarrow \mathscr{E}[\Lambda y_H.\texttt{hs}\ k_H\ f_S]$$

Figure 12: Haskell-Scheme operational semantics

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

Figure 13: ML syntax and evaluation contexts

$$\frac{}{\vdash_M \mathtt{L}} \qquad \frac{}{\vdash_M \mathtt{N}} \qquad \frac{}{\Gamma, y_M \vdash_M y_M}$$

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

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

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

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

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

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

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

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

Figure 14: ML typing rules

$$
\begin{aligned}
&\mathscr{E}[(\lambda x_M : t_M.e_M)\ u_M]_M \to \mathscr{E}[e_M[u_M/x_M]]\\
&\mathscr{E}[(\Lambda y_M.e_M)\langle t_M\rangle]_M \to \mathscr{E}[e_M[b \diamond t_M/y_M]]\\
&\mathscr{E}[\texttt{fix}\ (\lambda x_M : t_M.e_M)]_M \to \mathscr{E}[e_M[\texttt{fix}\ (\lambda x_M : t_M.e_M)/x_M]]\\
&\mathscr{E}[+\ \overline{n}\ \overline{n}']_M \to \mathscr{E}[\overline{n+n'}]\\
&\mathscr{E}[-\ \overline{n}\ \overline{n}']_M \to \mathscr{E}[\overline{max(n-n',0)}]\\
&\mathscr{E}[\texttt{if0}\ \overline{0}\ e_M\ e'_M]_M \to \mathscr{E}[e_M]\\
&\mathscr{E}[\texttt{if0}\ \overline{n}\ e_M\ e'_M]_M \to \mathscr{E}[e'_M]\ (n \neq 0)\\
&\mathscr{E}[\texttt{hd}\ (\texttt{nil}\ t_M)]_M \to \mathscr{E}[\texttt{wrong}\ t_M\ \text{“Empty list”}]\\
&\mathscr{E}[\texttt{tl}\ (\texttt{nil}\ t_M)]_M \to \mathscr{E}[\texttt{wrong}\ \{t_M\}\ \text{“Empty list”}]\\
&\mathscr{E}[\texttt{hd}\ (\texttt{cons}\ u_M\ u'_M)]_M \to \mathscr{E}[u_M]\\
&\mathscr{E}[\texttt{tl}\ (\texttt{cons}\ u_M\ u'_M)]_M \to \mathscr{E}[u'_M]\\
&\mathscr{E}[\texttt{null?}\ (\texttt{nil}\ t_M)]_M \to \mathscr{E}[\overline{0}]\\
&\mathscr{E}[\texttt{null?}\ (\texttt{cons}\ u_M\ u'_M)]_M \to \mathscr{E}[\overline{1}]\\
&\mathscr{E}[\texttt{wrong}\ t_M\ \mathit{string}]_H \to \textbf{Error: } \mathit{string}
\end{aligned}
$$

Figure 15: ML operational semantics

$$\mathscr{E}[\texttt{mh}\ t_M\ \texttt{L}\ (\texttt{hm}\ \texttt{L}\ t'_M\ f_M)]_M \to \mathscr{E}[f_M]\ (t_M = t'_M \text{ and } t_M \neq \texttt{L})$$
$$\mathscr{E}[\texttt{mh}\ t_M\ \texttt{L}\ (\texttt{hm}\ \texttt{L}\ t'_M\ f_M)]_M \to \mathscr{E}[\texttt{wrong}\ t_M\ \text{“Type mismatch”}]\ (t_M \neq t'_M \text{ and } t_M \neq \texttt{L})$$
$$\mathscr{E}[\texttt{mh}\ t_M\ \texttt{L}\ (\texttt{hs}\ \texttt{L}\ f_S)]_H \to \mathscr{E}[\texttt{wrong}\ t_M\ \text{“Bad value”}]\ (t_M \neq \texttt{L})$$
$$\mathscr{E}[\texttt{mh}\ \texttt{N}\ \texttt{N}\ \overline{n}]_M \to \mathscr{E}[\overline{n}]$$
$$\mathscr{E}[\texttt{mh}\ \{t_M\}\ \{t_H\}\ (\texttt{nil}\ t'_H)]_M \to \mathscr{E}[\texttt{nil}\ t_M]$$
$$\mathscr{E}[\texttt{mh}\ \{t_M\}\ \{t_H\}\ (\texttt{cons}\ e_H\ e'_H)]_M \to \mathscr{E}[\texttt{cons}\ (\texttt{mh}\ t_M\ t_H\ e_H)\ (\texttt{mh}\ \{t_M\}\ \{t_H\}\ e'_H)]$$
$$\mathscr{E}[\texttt{mh}\ (t_M \to t'_M)\ (t_H \to t'_H)\ (\lambda x_H : t''_H.e_H)]_M \to$$
$$\quad \mathscr{E}[\lambda x_M : t_M.\texttt{mh}\ t'_M\ t'_H\ ((\lambda x_H : t''_H.e_H)\ (\texttt{hm}\ t_H\ t_M\ x_M))]$$
$$\mathscr{E}[\texttt{mh}\ (\forall y_M.t_M)\ (\forall y_H.t_H)\ (\Lambda y'_H.e_H)]_M \to \mathscr{E}[\Lambda y_M.\texttt{mh}\ t_M\ t_H[\texttt{L}/y_H]\ e_H[\texttt{L}/y'_H]]$$

Figure 16: ML-Haskell operational semantics

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

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

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

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

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

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

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

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

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

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

Figure 17: ML-Scheme operational semantics

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

Figure 18: Scheme syntax and evaluation contexts

$$\frac{}{\vdash_S \texttt{TST}}$$

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

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

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

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

$$\frac{\Gamma \vdash_S e_S : \texttt{TST} \quad \Gamma \vdash_S e'_S : \texttt{TST} \quad \Gamma \vdash_S e''_S : \texttt{TST}}{\Gamma \vdash_S \texttt{if0}\ e_S\ e'_S\ e''_S : \texttt{TST}} \qquad \frac{}{\vdash_S \texttt{wrong}\ \mathit{string} : \texttt{TST}}$$

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

Figure 19: Scheme typing rules

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

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

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

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

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

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

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

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

$\mathscr{E}[c\ \texttt{nil}]_S \to \mathscr{E}[\texttt{wrong}\ \text{“Empty list”}]$

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

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

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

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

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

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

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

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

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

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

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

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

$\mathscr{E}[\texttt{wrong}\ \textit{string}]_S \to$ **Error:** *string*

Figure 20: Scheme operational semantics

$$\mathscr{E}[\texttt{sh}\ \texttt{L}\ (\texttt{hm}\ \texttt{L}\ k_M\ f_M)]_S \to \mathscr{E}[\texttt{wrong}\ \text{“Bad value”}]$$
$$\mathscr{E}[\texttt{sh}\ \texttt{L}\ (\texttt{hs}\ \texttt{L}\ f_S)]_S \to \mathscr{E}[f_S]$$
$$\mathscr{E}[\texttt{sh}\ \texttt{N}\ \overline{n}]_S \to \mathscr{E}[\overline{n}]$$
$$\mathscr{E}[\texttt{sh}\ \{k_H\}\ (\texttt{nil}\ t_H)]_S \to \mathscr{E}[\texttt{nil}]$$
$$\mathscr{E}[\texttt{sh}\ \{k_H\}\ (\texttt{cons}\ e_H\ e'_H)]_S \to \mathscr{E}[\texttt{cons}\ (\texttt{sh}\ k_H\ e_H)\ (\texttt{sh}\ \{k_H\}\ e'_H)]$$
$$\mathscr{E}[\texttt{sh}\ (k_H \to k'_H)\ (\lambda x_H : t_H.e_H)]_S \to$$
$$\qquad \mathscr{E}[\lambda x_S.\texttt{sh}\ k'_H\ ((\lambda x_H : t_H.e_H)\ (\texttt{hs}\ k_H\ x_S))]$$
$$\mathscr{E}[\texttt{sh}\ (\forall u_H.k_H)\ (\Lambda y'_H.e_H)]_S \to \mathscr{E}[\texttt{sh}\ k_H[\texttt{L}/u_H]\ e_H[\texttt{L}/y'_H]]$$

Figure 21: Scheme-Haskell operational semantics

$$\mathscr{E}[\texttt{sm}\,\texttt{L}\ (\texttt{mh}\,\texttt{L}\ k_H\ e_H)]_S \rightarrow \mathscr{E}[\texttt{wrong}\ \text{“Bad value”}]$$

$$\mathscr{E}[\texttt{sm}\,\texttt{L}\ (\texttt{ms}\,\texttt{L}\ f_S)]_S \rightarrow \mathscr{E}[f_S]$$

$$\mathscr{E}[\texttt{sm}\,\texttt{N}\ \overline{n}]_S \rightarrow \mathscr{E}[\overline{n}]$$

$$\mathscr{E}[\texttt{sm}\ \{k_M\}\ (\texttt{nil}\ t_M)]_S \rightarrow \mathscr{E}[\texttt{nil}]$$

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

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

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

Figure 22: Scheme-ML operational semantics

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

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] 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.