Interoperation for Lazy and Eager Evaluation

- Matthews & Findler
  - New method of interoperation
  - Type safety, observational equivalence & transparency
  - Eager evaluation strategies
- Lazy vs. eager

Lambda calculus

Typing

Model

Interoperation

Laziness

Solution

Syntax

Expressions, e	(1)	$x \in e$
$v = \lambda\, x\, .\, e$	(2)	$M \in e \Rightarrow \lambda\, x\, .\, M \in e$
$e = x \mid v \mid e\ e$	(3)	$M, N \in e \Rightarrow M\ N \in e$

(1) x
(2) $\lambda\ x\ .\ e$
(3) $e\ e$

z
by (1)

$\lambda\ z\ .\ z$
by (1), (2)

$(\lambda\ z\ .\ z)\ (\lambda\ z\ .\ z)$
by (1), (2), (3)

Syntactic sugar

$$\lambda\, x\,.\, x\; x \equiv \lambda\, x\,.\,(x\; x) \neq (\lambda\, x\,.\, x)\; x$$

$$\lambda\, x\; x'\,.\,\mathrm{e} \equiv \lambda\, x\,.\,\lambda\, x'\,.\,\mathrm{e}$$

$$\mathrm{e}\;\mathrm{e}'\;\mathrm{e}'' \equiv (\mathrm{e}\;\mathrm{e}')\;\mathrm{e}''$$

Semantics

Reduction relation, $\rightarrow$

$$e \rightarrow e'$$

$$e' \rightarrow e''$$

$$e \rightarrow e' \rightarrow e''$$

$$(\lambda x\,.\,e)\ v \rightarrow e\,[\,v\,/\,x\,]$$

$$e \rightarrow e' \Rightarrow e\ e'' \rightarrow e'\ e''$$

$$\frac{e \rightarrow e'}{e\ e'' \rightarrow e'\ e''}$$

$$e \rightarrow e' \Rightarrow v\ e \rightarrow v\ e'$$

$$\frac{e \rightarrow e'}{v\ e \rightarrow v\ e'}$$

term [*term argument* / *term parameter*] = *term*′

$$x\,[\,e\,/\,x\,] \;=\; e$$

$$x\,[\,e\,/\,x'\,] \;=\; x$$

$$(\lambda\, x\,.\,e)\,[\,e'\,/\,x\,] \;=\; \lambda\, x\,.\,e$$

$$(\lambda\, x\,.\,e)\,[\,e'\,/\,x'\,] \;=\; \lambda\, x\,.\,(e\,[\,e'\,/\,x'\,])$$

$$(e\; e')\,[\,e''\,/\,x\,] \;=\; (e\,[\,e''\,/\,x\,])\;(e'\,[\,e''\,/\,x\,])$$

$\ldots y \ldots$	$\lambda y . (\ldots y \ldots)$
Free variable	*Bound variable*
y	$\lambda y . y$
Open term	*Closed term*

Errors

$$\textit{error condition} \to \textit{error}$$

$$\frac{e \to e'}{e\ e'' \to e'\ e''}$$

$$\frac{e \to \textit{error}}{e\ e' \to \textit{error}}$$

$$\frac{e \to e'}{v\ e \to v\ e'}$$

$$\frac{e \to \textit{error}}{v\ e \to \textit{error}}$$

Evaluation contexts, *E*

$$E = [\ ] \mid E\ \mathrm{e} \mid v\ E$$

$$E' = \ldots[\ ]\ldots$$

$$E'\ [\ \mathrm{e}\ ] = \ldots\mathrm{e}\ldots$$

$$E\ [\ (\lambda\ x\ .\ \mathrm{e})\ v\ ] \rightarrow E\ [\ \mathrm{e}\ [\ v\ /\ x\ ]\ ]$$

$$E\ [\ \textit{error condition}\ ] \rightarrow \textit{error}$$

$$v \quad = \quad \lambda\, x\, .\, \mathrm{e} \mid \underline{n}$$

$$\mathrm{e} \quad = \quad x \mid v \mid \mathrm{e}\ \mathrm{e} \mid \textbf{+/-}\ \mathrm{e}\ \mathrm{e} \mid \textbf{if0}\ \mathrm{e}\ \mathrm{e}\ \mathrm{e} \mid \textbf{fun?}\ \mathrm{e}$$
$$\textbf{num?}\ \mathrm{e} \mid \textbf{wrong}\ \textit{string}$$

$$E \quad = \quad [\,] \mid E\ \mathrm{e} \mid v\ E \mid \textbf{+/-}\ E\ \mathrm{e} \mid \textbf{+/-}\ v\ E \mid \textbf{if0}\ E\ \mathrm{e}\ \mathrm{e}$$
$$\textbf{fun?}\ E \mid \textbf{num?}\ E$$

E [**wrong** *string*] → **Error:** *string*

$E[\ \mathbf{+}\ \underline{n}\ \underline{n'}\ ] \rightarrow E[\ \underline{n + n'}\ ]$

$E[\ \mathbf{-}\ \underline{n}\ \underline{n'}\ ] \rightarrow E[\ \underline{\max(n - n', 0)}\ ]$

$E[\ \mathbf{if0}\ \underline{0}\ \mathrm{e}\ \mathrm{e'}\ ] \rightarrow E[\ \mathrm{e}\ ]$
$E[\ \mathbf{if0}\ \underline{n}\ \mathrm{e}\ \mathrm{e'}\ ] \rightarrow E[\ \mathrm{e'}\ ]$

$E[\ \mathbf{fun?}\ (\lambda\ x\ .\ \mathrm{e})\ ] \rightarrow E[\ \underline{0}\ ]$
$E[\ \mathbf{fun?}\ v\ ] \rightarrow E[\ \underline{1}\ ]$

$E[\ \mathbf{num?}\ \underline{n}\ ] \rightarrow E[\ \underline{0}\ ]$
$E[\ \mathbf{num?}\ v\ ] \rightarrow E[\ \underline{1}\ ]$

Lambda calculus
Typing
Model
Interoperation
Laziness
Solution

Simple types

Types, *t*

$$t = \mathbf{N} \mid t \to t$$

$$\lambda\, x : t \,.\, e$$

$$t \to t \to t \equiv t \to (t \to t)$$

Judgments, $\vdash$

$$e : t \equiv (e , t)$$

$$\Gamma \equiv x_n : t_n , \ldots , x_I : t_I$$

$$\lambda\, x_n : t_n . (\ldots\, \lambda\, x_I : t_I . e)$$

$$\Gamma \vdash e : t \qquad \vdash e : t$$

$$\Gamma \vdash t \qquad \vdash t$$

Number type

$$\vdash \mathbf{N}$$

Function type

$$\frac{\Gamma \vdash t \quad \Gamma \vdash t'}{\Gamma \vdash t \to t'}$$

Number

$$\vdash \underline{n} : \mathbf{N}$$

Variable

$$\Gamma, x : t \vdash x : t$$

Function

$$\frac{\Gamma, x : t \vdash e : t'}{\Gamma \vdash \lambda\, x : t \,.\, e : t \to t'}$$

Application

$$\frac{\Gamma \vdash e : t \to t' \quad \Gamma \vdash e' : t}{\Gamma \vdash e\ e' : t'}$$

Arithmetic

$$\frac{\Gamma \vdash e : \mathbf{N} \quad \text{—} \quad \Gamma \vdash e' : \mathbf{N}}{\Gamma \vdash \mathbf{+/-}\ e\ e' : \mathbf{N}}$$

Condition

$$\frac{\Gamma \vdash e : \mathbf{N} \quad \text{—} \quad \Gamma \vdash e'/e'' : t}{\Gamma \vdash \mathbf{if0}\ e\ e'\ e'' : t}$$

Error

$$\frac{\Gamma \vdash t}{\Gamma \vdash \mathbf{wrong}\ t\ \mathit{string} : t}$$

Closed-term typing

Type

$$\Gamma \vdash \mathbf{T}$$

Number

$$\Gamma \vdash \underline{n} : \mathbf{T}$$

Variable

$$\Gamma, x : \mathbf{T} \vdash x : \mathbf{T}$$

Application

$$\frac{\Gamma \vdash e : \mathbf{T} \quad \text{—} \quad \Gamma \vdash e' : \mathbf{T}}{\Gamma \vdash e\ e' : \mathbf{T}}$$

Function

$$\frac{\Gamma, x : \mathbf{T} \vdash e : \mathbf{T}}{\Gamma \vdash \lambda\, x \,.\, e : \mathbf{T}}$$

Arithmetic

$$\frac{\Gamma \vdash e : \mathbf{T} \quad — \quad \Gamma \vdash e' : \mathbf{T}}{\Gamma \vdash \mathbf{+/-}\ e\ e'}$$

Predicate

$$\frac{\Gamma \vdash e : \mathbf{T}}{\Gamma \vdash \textbf{fun?/num?}\ e : \mathbf{T}}$$

Error

$$\Gamma \vdash \textbf{wrong}\ \textit{string} : \mathbf{T}$$

Type abstraction

Type variable

y

Universally-quantified types

$\forall\, y\, .\, t$

Type abstraction

$\Lambda\, y\, .\, e$

Free & bound type variables

$\Lambda\, y\, .\, (\ldots y \ldots)$

Type application

$e \langle\, t\, \rangle$

Syntactic sugar

$$\Lambda\, y\, y' \,.\, \mathrm{e} \equiv \Lambda\, y \,.\, \Lambda\, y' \,.\, \mathrm{e}$$

$$\mathrm{e} \langle\, t \,\rangle \langle\, t' \,\rangle \equiv (\mathrm{e} \langle\, t \,\rangle) \langle\, t' \,\rangle$$

Semantics

$$E\,[\,(\Lambda\, y\,.\,e)\,\langle\, t\,\rangle\,] \rightarrow E\,[\,e\,[\,t\,/\,y\,]\,]$$

Type-in-term substitution

term [*type argument* / *type parameter*] = *term′*

$$x\,[\,t\,/\,y\,] \;=\; x$$

$$(\lambda\, x : t\,.\,\mathrm{e})\,[\,t\,/\,y\,] \;=\; \lambda\, x : t\,[\,t\,/\,y\,]\,.\,\mathrm{e}\,[\,t\,/\,y\,]$$

$$(\mathrm{e}\;\mathrm{e}')\,[\,t\,/\,y\,] \;=\; (\mathrm{e}\,[\,t\,/\,y\,])\;(\mathrm{e}'\,[\,t\,/\,y\,])$$

$$(\mathbf{+/-}\;\mathrm{e}\;\mathrm{e}')\,[\,t\,/\,y\,] \;=\; \mathbf{+/-}\;(\mathrm{e}\,[\,t\,/\,y\,])\;(\mathrm{e}'\,[\,t\,/\,y\,])$$

$$(\textbf{if0}\ e\ e'\ e'')\,[\,t\,/\,y\,] \;=\; \begin{array}{l}\textbf{if0}\ (e\,[\,t\,/\,y\,])\\ \quad\ (e'\,[\,t\,/\,y\,])\\ \quad\ (e''\,[\,t\,/\,y\,])\end{array}$$

$$(\Lambda\, y\,.\,e)\,[\,t\,/\,y\,] \;=\; \Lambda\, y\,.\,e$$

$$(\Lambda\, y\,.\,e)\,[\,t\,/\,y'\,] \;=\; \Lambda\, y\,.\,e\,[\,t\,/\,y'\,]$$

$$(e\,\langle\, t\,\rangle)\,[\,t'\,/\,y\,] \;=\; (e\,[\,t'\,/\,y\,])\,\langle\, t\,[\,t'\,/\,y\,]\,\rangle$$

Type-in-type substitution

type [type argument / type parameter] = type′

$$\mathbf{N}\,[\,t \,/\, y\,] \;=\; \mathbf{N}$$

$$(t \to t')\,[\,t \,/\, y\,] \;=\; t\,[\,t \,/\, y\,] \to t'\,[\,t \,/\, y\,]$$

$$y\,[\,t \,/\, y\,] \;=\; t$$

$$y\,[\,t \,/\, y'\,] \;=\; y$$

$$(\forall y\,.\,t)\,[\,t'\,/\,y\,] \;=\; \forall y\,.\,t$$

$$(\forall y\,.\,t)\,[\,t'\,/\,y'\,] \;=\; \forall y\,.\,t\,[\,t'\,/\,y'\,]$$

Lambda calculus
Typing
Model
Interoperation
Laziness
Solution

“ML” & “Scheme”

- Numbers, arithmetic & conditions
- Function abstractions & applications
- Errors
- Eager evaluation
- Language subscripts *m* and *s* for nonterminals

ML

- Statically typed
- Parametric polymorphism

ML syntax

$$t_m \;=\; \mathbf{N} \mid y_m \mid t_m \to t_m \mid \forall\, y_m \,.\, t_m$$

$$v_m \;=\; \underline{n} \mid \lambda\, x_m : t_m \,.\, e_m \mid \Lambda\, y_m \,.\, e_m$$

$$e_m \;=\; x_m \mid v_m \mid e_m\; e_m \mid e_m \langle\, t_m \,\rangle \mid \mathbf{+/-}\; e_m\; e_m$$
$$\mathbf{if0}\; e_m\; e_m\; e_m \mid \mathbf{wrong}\; t_m\; \mathit{string}$$

$$E_m \;=\; [\;]_m \mid E_m\; e_m \mid v_m\; E_m \mid E_m \langle\, t_m \,\rangle$$
$$\mathbf{+/-}\; E_m\; e_m \mid \mathbf{+/-}\; v_m\; E_m \mid \mathbf{if0}\; E_m\; e_m\; e_m$$

Scheme

- Dynamically typed
- Closed-term typing
- Ad-hoc polymorphism

Scheme syntax

v_s = $\underline{n}$ | $\lambda\, x_s\, .\, e_s$

e_s = x_s | v_s | $e_s\ e_s$ | **+/-** $e_s\ e_s$ | **if0** $e_s\ e_s\ e_s$ | **fun?** e_s
num? e_s | **wrong** *string*

E_s = $[\ ]_s$ | $E_s\ e_s$ | $v_s\ E_s$ | **+/-** $E_s\ e_s$ | **+/-** $v_s\ E_s$
if0 $E_s\ e_s\ e_s$ | **fun?** E_s | **num?** E_s

Lambda calculus
Typing
Model
Interoperation
Laziness
Solution

Syntax

Converting from Scheme to ML

$$e_m = \cdots \mid \mathbf{ms}\ t_m\ e_s$$

$$E_m = \cdots \mid \mathbf{ms}\ t_m\ E_s$$

Converting from ML to Scheme

$$e_s = \cdots \mid \mathbf{sm}\ t_m\ e_m$$

$$E_s = \cdots \mid \mathbf{sm}\ t_m\ E_m$$

Typing

ML–Scheme

$$\frac{\Gamma \vdash_m t_m \quad \text{—} \quad \Gamma \vdash_s e_s : \mathbf{T}}{\Gamma \vdash_m \mathbf{ms}\ t_m\ e_s : t_m}$$

Scheme–ML

$$\frac{\Gamma \vdash_m t_m \quad \text{—} \quad \Gamma \vdash_m e_m : t_m' \quad \text{—} \quad t_m = t_m'}{\Gamma \vdash_s \mathbf{sm}\ t_m\ e_m : \mathbf{T}}$$

Converting numbers

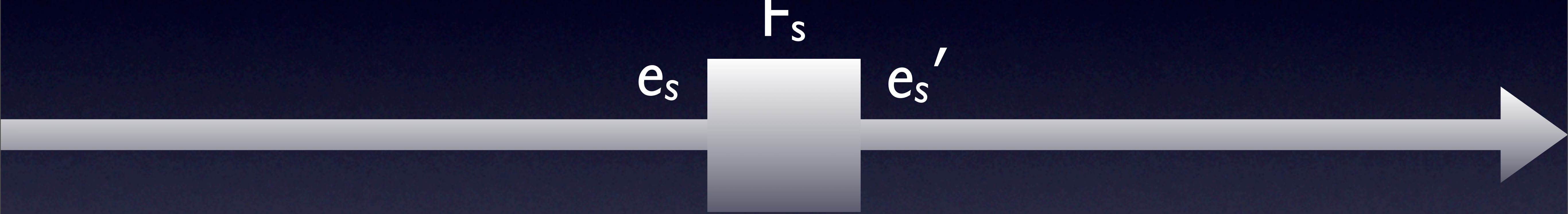
Outermost evaluation context, $\mathcal{E}$

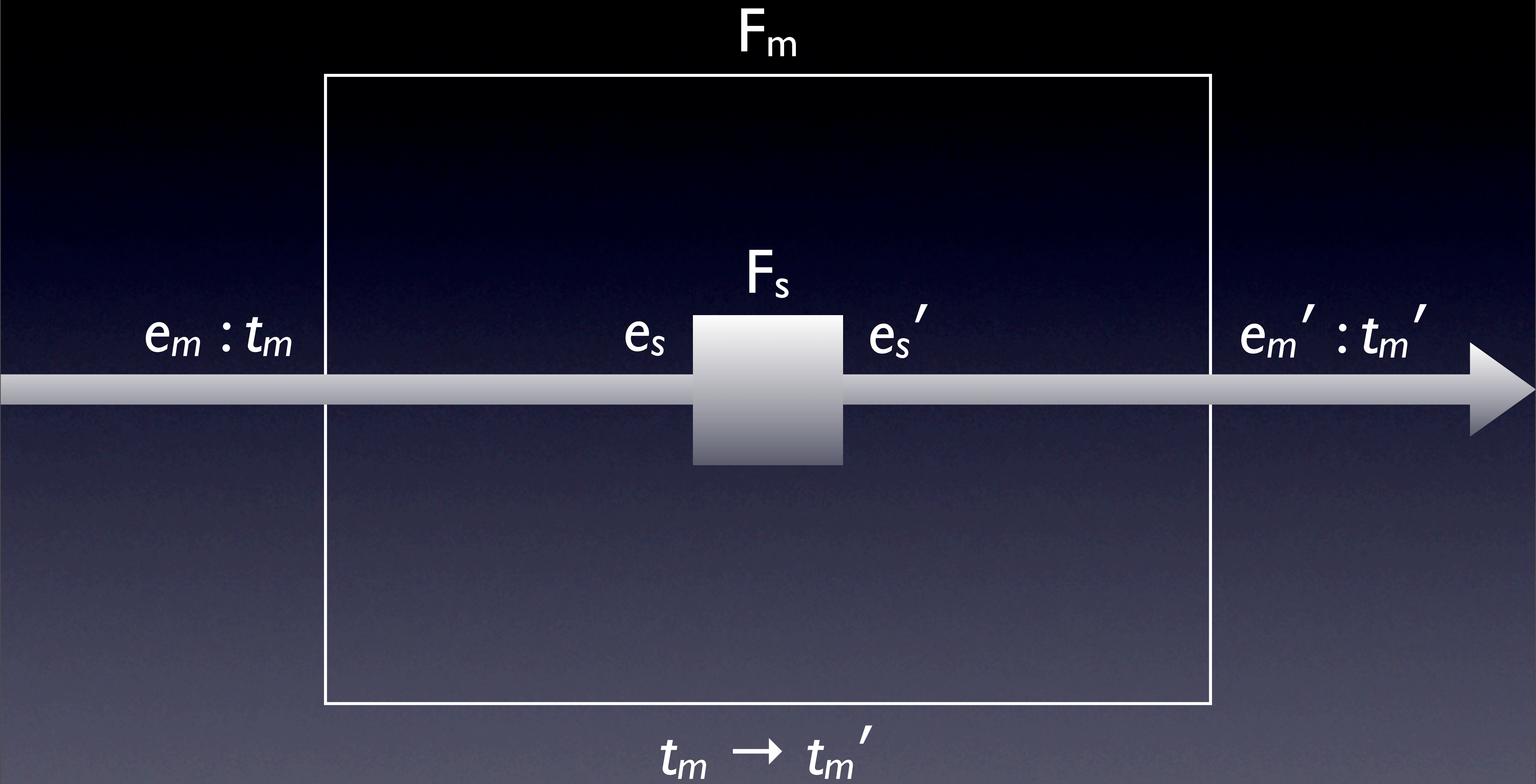
$$\mathcal{E}\,[\ \mathbf{ms}\ \mathbf{N}\ \underline{n}\ ]_m \to \mathcal{E}\,[\ \underline{n}\ ]$$

$$\mathcal{E}\,[\ \mathbf{sm}\ \mathbf{N}\ \underline{n}\ ]_s \to \mathcal{E}\,[\ \underline{n}\ ]$$

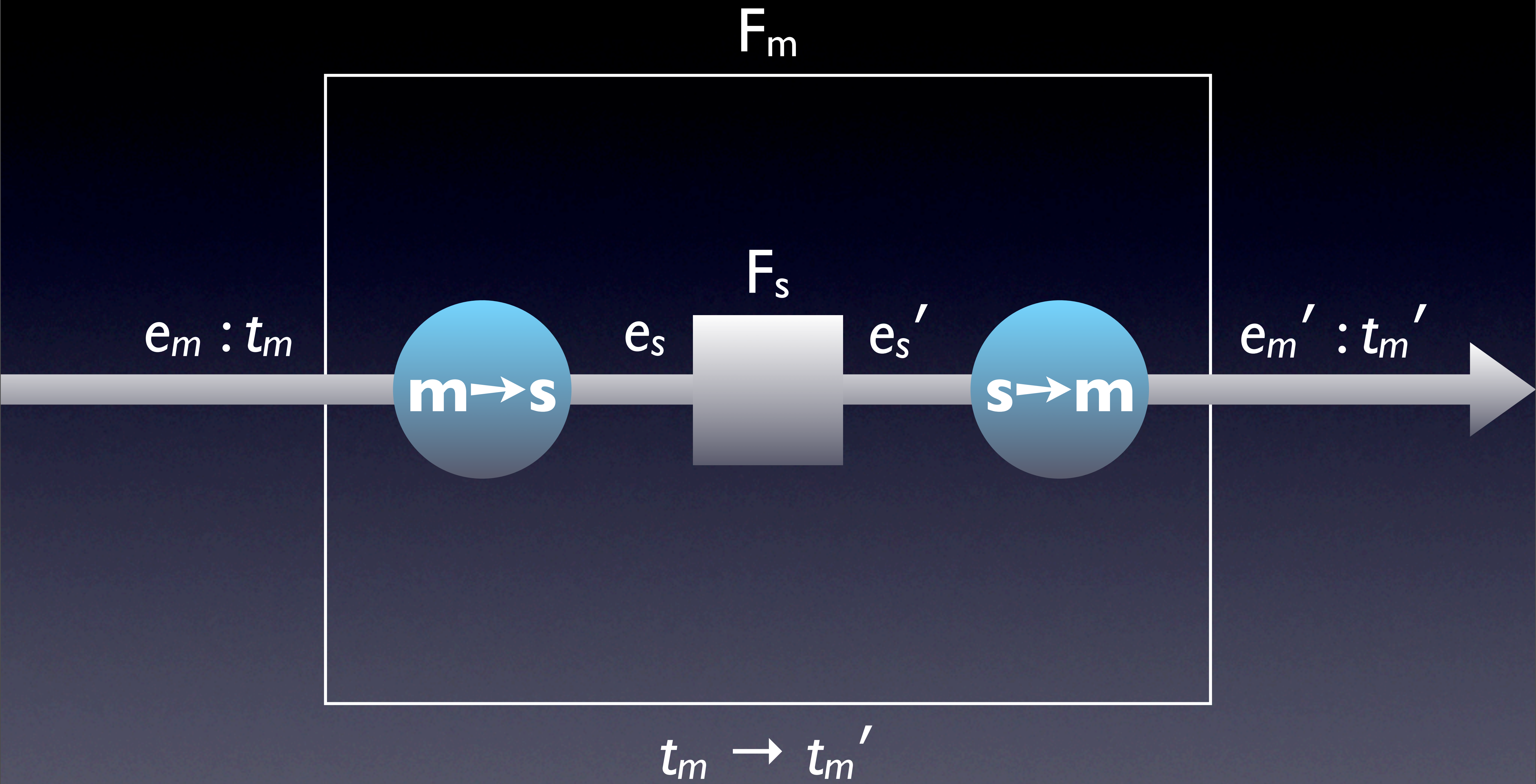
$$\mathcal{E}\,[\ \mathbf{ms}\ \mathbf{N}\ v_s\ ]_m \to \mathcal{E}\,[\ \mathbf{wrong}\ \mathbf{N}\ \text{“Not a number”}\ ]$$

Converting functions





F_m
F_s
$e_m : t_m$
e_s
e_s'
$e_m' : t_m'$
$t_m \rightarrow t_m'$


F_m
$e_m : t_m$
m→s
e_s
F_s
e_s'
s→m
$e_m' : t_m'$
$t_m \rightarrow t_m'$


F_m
$e_m : t_m$
$e_m' : t_m'$
$t_m \rightarrow t_m'$

Convert from ML to Scheme

$\mathrm{sm} \equiv \mathbf{sm}\ t_m\ x_m$

Apply the Scheme function

$\mathrm{app} \equiv F_s\ \mathrm{sm}$

Convert from Scheme to ML

$\mathrm{ms} \equiv \mathbf{ms}\ t_m'\ \mathrm{app}$

Abstract the ML argument

$F_m \equiv \lambda\ x_m : t_m\ .\ \mathrm{ms}$

Converted Scheme function in ML

$F_m \equiv \lambda\ x_m : t_m\ .\ \mathbf{ms}\ t_m'\ (F_s\ (\mathbf{sm}\ t_m\ x_m))$

$$\mathcal{E}[\ \mathbf{ms}\ (t_m \to t_m{}')\ (\lambda\ x_s\ .\ e_s)\ ]_m$$

$$\to$$

$$\mathcal{E}[\ \lambda\ x_m : t_m\ .\ \mathbf{ms}\ t_m{}'\ ((\lambda\ x_s\ .\ e_s)\ (\mathbf{sm}\ t_m\ x_m))\ ]$$

$$\mathcal{E}[\ \mathbf{sm}\ (t_m \to t_m{}')\ v_m\ ]_s$$

$$\to$$

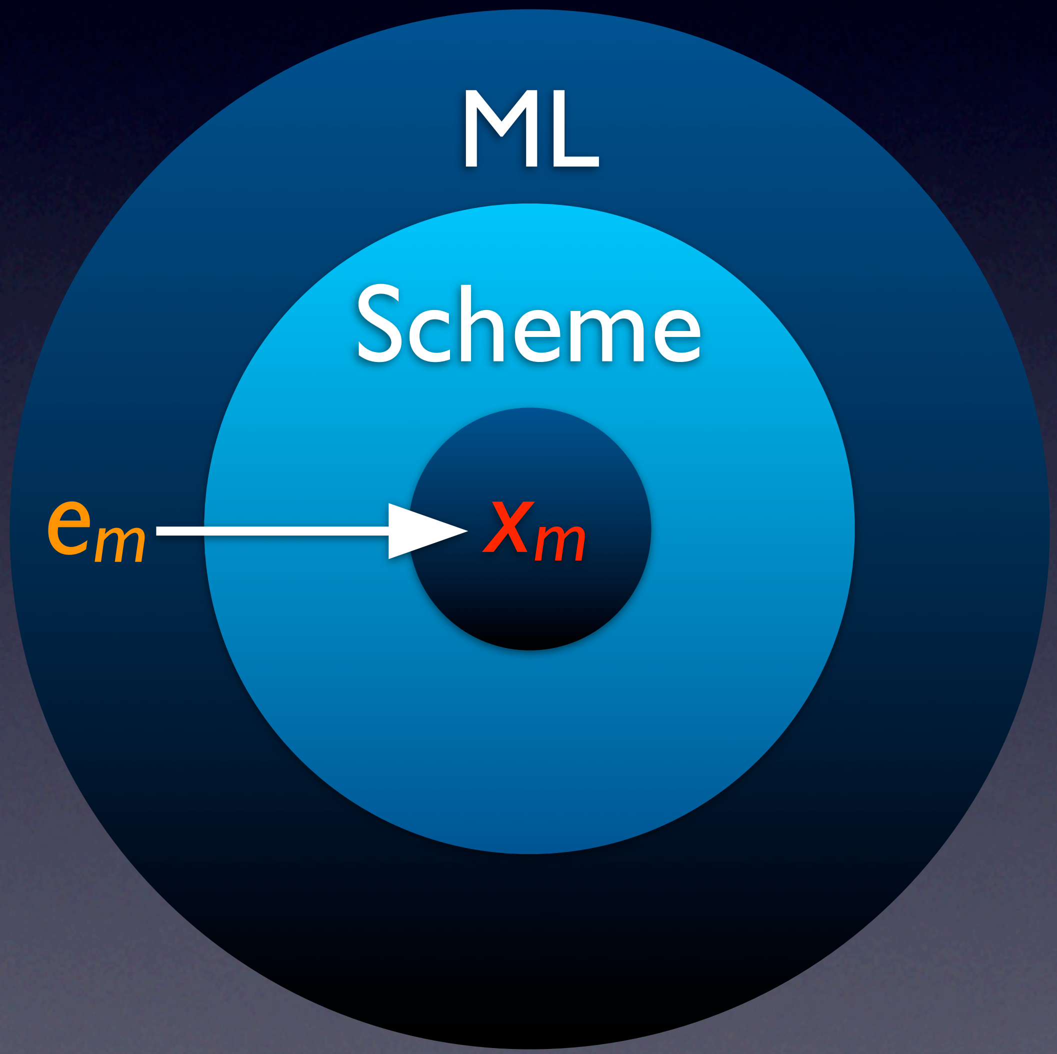
$$\mathcal{E}[\ \lambda\ x_s\ .\ \mathbf{sm}\ t_m{}'\ (v_m\ (\mathbf{ms}\ t_m\ x_s))\ ]$$

$$\mathcal{E}[\ \mathbf{ms}\ (t_m \to t_m{}')\ v_s\ ]_m$$

$$\to$$

$$\mathcal{E}[\ \mathbf{wrong}\ (t_m \to t_m{}')\ \text{“Not a function”}\ ]$$

$$(\mathbf{ms}\ t_m{}'\ ((\lambda\ x_s\ .\ e_s)\ (\mathbf{sm}\ t_m\ x_m)))\ [\ e_m\ /\ x_m\ ]$$



Boundary substitution

$$(\textbf{ms}\ t_m\ e_s)\ [\ e_m\ /\ x_m\ ] = \textbf{ms}\ t_m\ (e_s\ [\ e_m\ /\ x_m\ ])$$

Foreign substitution

$$(\ldots e_s \ldots)\ [\ e_m\ /\ x_m\ ] = \ldots e_s\ [\ e_m\ /\ x_m\ ] \ldots$$

$$(\textbf{sm}\ t_m\ e_m)\ [\ e_m'\ /\ x_m\ ] = \textbf{sm}\ t_m\ (e_m\ [\ e_m'\ /\ x_m\ ])$$

Converting type abstractions

$t_m = \cdots \mid \mathbf{L}$ $\Gamma \vdash_m \mathbf{L}$ $v_m = \cdots \mid \mathbf{ms}\ \mathbf{L}\ v_s$

$$\mathcal{E}\ [\ \mathbf{sm}\ (\forall\ y_m\ .\ t_m)\ (\Lambda\ y_m'\ .\ e_m)\ ]_s$$

$$\rightarrow$$

$$\mathcal{E}\ [\ \mathbf{sm}\ t_m\ [\ \mathbf{L}\ /\ y_m\ ]\ e_m\ [\ \mathbf{L}\ /\ y_m'\ ]\ ]$$

$$\mathcal{E}\ [\ \mathbf{sm}\ \mathbf{L}\ (\mathbf{ms}\ \mathbf{L}\ v_s)\ ]_s \rightarrow \mathcal{E}\ [\ v_s\ ]$$

$$\mathrm{id} \equiv \Lambda\, y \,.\, \mathbf{ms}\ (y \rightarrow y)\ (\lambda\, x \,.\, x)$$

$$\mathrm{id}\ \langle\ \mathbf{N}\ \rangle$$

behaves the same as

$$\mathrm{id}\ \langle\ \mathbf{N} \rightarrow \mathbf{N}\ \rangle$$

$$\mathrm{id}_m \equiv \Lambda\, y \,.\, \mathbf{ms}\ (y \rightarrow y)\ (\lambda\, x \,.\, \mathbf{if0}\ (\mathbf{num?}\ x)\ x\ \underline{0})$$

$$\mathrm{id}_m\ \langle\ \mathbf{N}\ \rangle$$

behaves differently than

$$\mathrm{id}_m\ \langle\ \mathbf{N} \rightarrow \mathbf{N}\ \rangle$$

Conversion schemes

$$id_{good} \equiv \Lambda\, y\, .\, \mathbf{ms}\ (y \rightarrow y)\ (\lambda\, x\, .\, x)$$

$$id_{good} \langle\ \mathbf{N}\ \rangle$$

behaves the same as

$$id_{good} \langle\ \mathbf{N} \rightarrow \mathbf{N}\ \rangle$$

$$id_{bad} \equiv \Lambda\, y\, .\, \mathbf{ms}\ (y \rightarrow y)\ (\lambda\, x\, .\, \mathbf{if0}\ (\mathbf{num?}\ x)\ x\ \underline{0})$$

$$id_{bad} \langle\ \mathbf{N}\ \rangle$$

behaves differently than

$$id_{bad} \langle\ \mathbf{N} \rightarrow \mathbf{N}\ \rangle$$

Conversion schemes, *k*

$$k_m = \mathbf{L} \mid \mathbf{N} \mid y_m \mid k_m \rightarrow k_m \mid \forall\, y_m \,.\, k_m \mid b \diamond t_m$$

$$e_m = \cdots \mid \mathbf{ms}\ k_m\ e_s$$

$$e_s = \cdots \mid \mathbf{sm}\ k_m\ e_m$$

$$v_s = \cdots \mid \mathbf{sm}\ k_m\ v_m$$

$$\frac{\Gamma \vdash_m \lfloor k_m \rfloor \quad \Gamma \vdash_s e_s : \mathbf{T}}{\Gamma \vdash_m \mathbf{ms}\ k_m\ e_s : \lfloor k_m \rfloor}$$

$$\mathcal{E}\,[\,(\Lambda\, y_m\,.\,e_m)\,\langle\, t_m\,\rangle\,]_m \rightarrow \mathcal{E}\,[\,e_m\,[\,b \diamond t_m\,/\,y_m\,]\,]$$

$$\mathcal{E}\,[\,\mathbf{ms}\,(b \diamond t_m)\,(\mathbf{sm}\,(b \diamond t_m)\,v_m)\,]_m \rightarrow \mathcal{E}\,[\,v_m\,]$$

$$\mathcal{E}\,[\,\mathbf{ms}\,(b \diamond t_m)\,v_s\,]_m$$

$$\rightarrow$$

$$\mathcal{E}\,[\,\mathbf{wrong}\,\lfloor\, b \diamond t_m\,\rfloor\,\text{“Brand mismatch”}\,]$$

$$
\begin{aligned}
\lfloor b \diamond t_m \rfloor &= t_m \\
\lfloor \mathbf{L} \rfloor &= \mathbf{L} \\
\lfloor \mathbf{N} \rfloor &= \mathbf{N} \\
\lfloor y_m \rfloor &= y_m \\
\lfloor k_m \rightarrow k_m \rfloor &= \lfloor k_m \rfloor \rightarrow \lfloor k_m \rfloor \\
\lfloor \forall\, y_m \,.\, k_m \rfloor &= \forall\, y_m \,.\, \lfloor k_m \rfloor
\end{aligned}
$$

$$(\lambda\ x_m : t_m\ .\ e_m)\ [\ b \diamond t_m{}'\ /\ y_m\ ]$$

$$=$$

$$\lambda\ x_m : t_m\ [\ t_m{}'\ /\ y_m\ ]\ .\ e_m\ [\ b \diamond t_m{}'\ /\ y_m\ ]$$

$$(\mathbf{ms}\ k_m\ e_s)\ [\ b \diamond t_m{}'\ /\ y_m\ ]$$

$$=$$

$$\mathbf{ms}\ k_m\ [\ b \diamond t_m{}'\ /\ y_m\ ]\ e_s\ [\ b \diamond t_m{}'\ /\ y_m\ ]$$

Lambda calculus
Typing
Model
Interoperation
Laziness
Solution

“Haskell”

- Eager vs. lazy evaluation
- Add “Haskell” to the mix
- Interoperation forces Haskell evaluation
  - Applying converted functions in Haskell
  - Converting values from Haskell
- Observational equivalence and transparency do not hold

Haskell syntax

$$t_h \;=\; \mathbf{N} \mid y_h \mid t_h \rightarrow t_h \mid \forall\, y_h \,.\, t_h$$

$$k_h \;=\; \mathbf{L} \mid \mathbf{N} \mid y_h \mid k_h \rightarrow k_h \mid \forall\, y_h \,.\, k_h \mid b \diamond t_h$$

$$v_h \;=\; \underline{n} \mid \lambda\, x_h : t_h \,.\, \mathrm{e}_h \mid \Lambda\, y_h \,.\, \mathrm{e}_h \mid \mathbf{hs\ L}\ v_s$$

$$\mathrm{e}_h \;=\; \begin{array}{l} x_h \mid v_h \mid \mathrm{e}_h\ \mathrm{e}_h \mid \mathrm{e}_h \langle\, t_h \,\rangle \mid \mathbf{+/-}\ \mathrm{e}_h\ \mathrm{e}_h \\ \mathbf{if0}\ \mathrm{e}_h\ \mathrm{e}_h\ \mathrm{e}_h \mid \mathbf{wrong}\ t_h\ \mathit{string} \end{array}$$

$$E_h \;=\; \begin{array}{l} [\ ]_h \mid E_h\ \mathrm{e}_h \mid E_h \langle\, t_h \,\rangle \mid \mathbf{+/-}\ E_h\ \mathrm{e}_h \\ \mathbf{+/-}\ v_h\ E_h \mid \mathbf{if0}\ E_h\ \mathrm{e}_h\ \mathrm{e}_h \end{array}$$

Interoperation syntax

$$e_h = \cdots \mid \textbf{hm}\ t_h\ t_m\ e_m \mid \textbf{hs}\ k_h\ e_s$$

$$e_m = \cdots \mid \textbf{mh}\ t_m\ t_h\ e_h$$

$$e_s = \cdots \mid \textbf{sh}\ k_h\ e_h$$

$$v_h = \cdots \mid \textbf{hs}\ \textbf{L}\ v_s$$

$$v_s = \cdots \mid \textbf{sh}\ k_h\ e_h$$

$$E_h = \cdots \mid \mathbf{hm}\ t_h\ t_m\ E_m \mid \mathbf{hs}\ k_h\ E_s$$

$$E_m = \cdots \mid \mathbf{mh}\ t_m\ t_h\ E_h$$

$$E_s = \cdots \mid \mathbf{sh}\ k_h\ E_h$$

Applying functions

$$\text{zero}_s \equiv \lambda\, x\, .\, \underline{0}$$

$$\vdash_h \text{zero}_h : \mathbf{N} \to \mathbf{N}$$

$$\text{zero}_h \equiv \mathbf{hs}\ (\mathbf{N} \to \mathbf{N})\ \text{zero}_s$$

$$\text{broken} \equiv \mathbf{wrong}\ \mathbf{N}\ \text{“Broken”}$$

$$\text{zero}_h\ \text{broken} \to \underline{0}$$

$$\text{zero}_h\ \text{broken} \to^{*} \textbf{Error:}\ \text{“Broken”}$$

$zero_h$ broken ≡ (**hs** (**N** → **N**) $zero_s$) broken

→ (λ x : **N** . **hs N** ($zero_s$ (**sh N** x))) broken

→ **hs N** ($zero_s$ (**sh N** broken))

→ **Error:** “Broken”

Converting values

$e_h = \cdots \mid \textbf{nil}\ t_h \mid \textbf{cons}\ e_h\ e_h \mid \textbf{hd}\ e_h \mid \textbf{tl}\ e_h \mid \textbf{null?}\ e_h$

$t_h = \cdots \mid \{\ t_h\ \}$

$k_h = \cdots \mid \{\ k_h\ \}$

$E_h = \cdots \mid \textbf{hd}\ E_h \mid \textbf{tl}\ E_h \mid \textbf{null?}\ E_h$

$$\frac{\Gamma \vdash_h t_h}{\Gamma \vdash_h \{\, t_h \,\}}$$

$$\frac{\Gamma \vdash_h e_h : \{\, t_h \,\}}{\Gamma \vdash_h \mathbf{hd}\ e_h : t_h}$$

$$\frac{\Gamma \vdash_h t_h}{\Gamma \vdash_h \mathbf{nil}\ t_h : \{\, t_h \,\}}$$

$$\frac{\Gamma \vdash_h e_h : \{\, t_h \,\}}{\Gamma \vdash_h \mathbf{tl}\ e_h : \{\, t_h \,\}}$$

$$\frac{\Gamma \vdash_h e_h : \{\, t_h \,\}}{\Gamma \vdash_h \mathbf{null?}\ e_h : \mathbf{N}}$$

$$\frac{\Gamma \vdash_h e_h : t_h \quad \Gamma \vdash_h e_h' : \{\, t_h \,\}}{\Gamma \vdash_h \mathbf{cons}\ e_h\ e_h' : \{\, t_h \,\}}$$

$\mathcal{E}$ [**hd** (**nil** t_h)]$_h$ → $\mathcal{E}$ [**wrong** t_h "Empty list"]

$\mathcal{E}$ [**hd** (**cons** e_h e_h')]$_h$ → $\mathcal{E}$ [e_h]

$\mathcal{E}$ [**tl** (**nil** t_h)]$_h$ → $\mathcal{E}$ [**wrong** { t_h } "Empty list"]

$\mathcal{E}$ [**tl** (**cons** e_h e_h')]$_h$ → $\mathcal{E}$ [e_h']

$\mathcal{E}$ [**null?** (**nil** t_h)]$_h$ → $\mathcal{E}$ [$\underline{0}$]

$\mathcal{E}$ [**null?** (**cons** e_h e_h')]$_h$ → $\mathcal{E}$ [$\underline{1}$]

$$\mathscr{E}[\,\textbf{hs}\ \{\ t_h\ \}\ \textbf{nil}\,]_h \rightarrow \mathscr{E}[\,\textbf{nil}\ t_h\,]$$

$$\mathscr{E}[\,\textbf{hs}\ \{\ t_h\ \}\ (\textbf{cons}\ v_s\ v_s{}')\,]_h$$

$$\rightarrow$$

$$\mathscr{E}[\,\textbf{cons}\ (\textbf{hs}\ t_h\ v_s)\ (\textbf{hs}\ \{\ t_h\ \}\ v_s{}'\,]\,]$$

$$v_m = \cdots \mid \textbf{nil}\ t_m \mid \textbf{cons}\ v_m\ v_m$$

$$v_s = \cdots \mid \textbf{nil} \mid \textbf{cons}\ v_s\ v_s$$

$$E_m = \cdots \mid \textbf{cons}\ E_m\ \mathrm{e}_m \mid \textbf{cons}\ v_m\ E_m$$

$$E_s = \cdots \mid \textbf{cons}\ E_s\ \mathrm{e}_s \mid \textbf{cons}\ v_s\ E_s$$

$$\text{broken}_h \equiv \textbf{wrong N} \text{ “Broken”}$$

$$\text{empty}_h \equiv \textbf{nil N}$$

$$\text{list}_h \equiv \textbf{cons } \text{broken}_h \text{ empty}_h$$

$$\text{list}_s \equiv \textbf{sh } \{\textbf{N}\} \text{ list}_h$$

$$\text{list}_s \rightarrow \textbf{cons } \text{broken}_s \text{ empty}_s$$

$$\text{list}_s \rightarrow^* \textbf{Error:} \text{ “Broken”}$$

$\text{list}_s \equiv \underline{\textbf{sh } \{\textbf{N}\} \ (\textbf{cons } \text{broken}_h \ \text{empty}_h)}$

$\rightarrow$ **cons (sh N** $\underline{\text{broken}_h}$**) (sh {N}** empty_h**)**

$\rightarrow$ **Error:** "Broken"

Haskell & ML

$$\mathcal{E}\,[\;\mathbf{hm}\;(\forall\, y_h\,.\,t_h)\;(\forall\, y_m\,.\,t_m)\;(\Lambda\, y_m{}'\,.\,e_m)\;]_h$$

$$\rightarrow$$

$$\mathcal{E}\,[\;\Lambda\, y_h\,.\,\mathbf{hm}\;t_h\;(t_m\,[\,\mathbf{L}\,/\,y_m\,])\;(e_m\,[\,\mathbf{L}\,/\,y_m{}'\,])\;]$$

$$v_h = \cdots \mid \mathbf{hm}\;\mathbf{L}\;t_m\;v_m$$

$$v_m = \cdots \mid \mathbf{mh}\;\mathbf{L}\;t_h\;e_h$$

$$\frac{\Gamma \vdash_h t_h \;—\; \Gamma \vdash_m t_m \;—\; t_h \doteq t_m \;—\; \Gamma \vdash_m e_m : t_m' \;—\; t_m = t_m'}{\Gamma \vdash_h \mathbf{hm}\ t_h\ t_m\ e_m : t_h}$$

$$x \doteq x$$

$$x \doteq y \Rightarrow y \doteq x$$

$$x \doteq y \wedge y \doteq z \Rightarrow x \doteq z$$

$$t_h \doteq \mathbf{L}$$

$$t_m \doteq \mathbf{L}$$

$$t_h = t_m \Rightarrow t_h \doteq t_m$$

Lambda calculus

Typing

Model

Interoperation

Laziness

Solution

Problematic contexts

Applying converted functions in Haskell

hs N (zero$_s$ (**sh N** broken))	v_s E_s

Converting values from Haskell

cons (**sh N** broken$_h$) (…)	**cons** E_s e_s

$E_s =$

$[\ ]_s$	**num?** E_s
$E_s\ e_s$	**if0** $E_s\ e_s\ e_s$
$v_s\ E_s$	**cons** $E_s\ e_s$
+/- $E_s\ e_s$	**cons** $v_s\ E_s$
+/- $v_s\ E_s$	**hd** E_s
fun? E_s	**tl** E_s
list? E_s	**sh** $k_h\ E_h$
null? E_s	**sm** $k_m\ E_m$

Restrict forcing

Rename E_s to U_s (unforced)

$U_s = E_s$

Factor Scheme–Haskell out of U_s (forced)

$F_s = U_s \mid \mathbf{sh}\ k_h\ E_h$

Rename v_s to u_s (unforced)

$u_s = v_s$

Factor Scheme–Haskell out of u_s (forced)

$f_s = u_s \mid \mathbf{sh}\ k_h\ e_h$

Redefine U_s in terms of U_s, F_s, u_s, f_s

$$
U_s \quad = \quad
\begin{array}{c}
\vdots \\
U_s\ e_s \\
u_s\ U_s \\
\textbf{cons}\ U_s\ e_s \\
\textbf{cons}\ u_s\ U_s \\
\vdots
\end{array}
$$

Redefine U_s in terms of U_s, F_s, u_s, f_s

$$
U_s = \begin{array}{c} \vdots \\ F_s\ e_s \\ f_s\ U_s \\ \textbf{cons}\ U_s\ e_s \\ \textbf{cons}\ u_s\ U_s \\ \vdots \end{array}
$$

$$F_m = U_m \mid \textbf{mh}\ t_m\ t_h\ E_h$$

$$U_s =$$

$[\ ]_s$	**fun?** F_s	**cons** U_s e_s
F_s e_s	**list?** F_s	**cons** u_s U_s
f_s E_s	**null?** F_s	**hd** F_s
+/- F_s e_s	**num?** F_s	**tl** F_s
+/- f_s F_s	**if0** F_s e_s e_s	**sh** k_h E_h
		sm k_m E_m

Rename $\mathscr{E}$ to $\mathscr{F}$

$$\mathscr{F}[\ (\lambda\ x_m : t_m\ .\ e_m)\ u_m\ ]_m \rightarrow \mathscr{F}[\ e_m\ [u_m\ /\ x_m]\ ]$$

$$\mathscr{F}[\ \textbf{hd}\ (\textbf{cons}\ u_m\ u_m')\ ]_m \rightarrow \mathscr{F}[\ u_m\ ]$$

$$\mathscr{F}[\ \textbf{tl}\ (\textbf{cons}\ u_m\ u_m')\ ]_m \rightarrow \mathscr{F}[\ u_m'\ ]$$

$$\mathscr{F}[\ \textbf{null}\ (\textbf{cons}\ u_m\ u_m')\ ]_m \rightarrow \mathscr{F}[\ \bot\ ]$$

Solution

- For incompatible evaluation contexts
- Mirrors laziness for Haskell boundaries
- Controls the forcing of Haskell evaluation
- Restores observational equivalence and transparency

Summary

- Matthews & Findler
- Lazy vs. eager
- Observational equivalence and transparency
- Incompatible evaluation contexts
- Laziness mirrored in eager evaluation

Questions