The Meaning of Multilanguage Programs

Jacob Matthews
jacobm@cs.uchicago.edu
University of Chicago

Robby Findler
robbyp@cs.uchicago.edu
University of Chicago
How do languages vary?
How do languages vary?

differing type systems
How do languages vary?

differing evaluation rules
How do languages vary?

differing values
How do languages vary?

This talk: differing type systems
How do languages vary?

This talk: differing evaluation rules (a little)
How do languages vary?

This talk: differing values (a very little)
\[ e := v | (e \ e) | (+ \ e \ e) \]
\[ v := (\lambda \ (x) \ e) | \text{number} \]

\[ C := [ ] | (v \ C) | (e \ C) \]

\[ C[((\lambda \ (x) \ e) \ v)] \rightarrow C[e \ [x := v]] \]
\[ C[(+ \ n_1 \ n_2)] \rightarrow C[n_1+n_2] \]
\[
e := v \mid (e \ e) \mid (+ \ e \ e)
\]
\[
v := (\lambda (x : \tau) \ e) \mid \text{number}
\]
\[
\tau := \text{int} \mid (\tau \to \tau)
\]
\[
C := [] \mid (v \ C) \mid (C \ e)
\]
\[
C[((\lambda (x : \tau) \ e) \ v)] \to C[e \ [x := v]]
\]
\[
C[(+ n_1 \ n_2)] \to C[n_1 + n_2]
\]
\[ n \in \{0, 1, 2, \ldots\} \]

\[ \Gamma \quad n : \text{int} \quad \text{MLNUM} \]

\[ x : \tau \in \Gamma \]

\[ \Gamma \quad x : \tau \quad \text{MLVAR} \]

\[ \frac{[x := \tau_A] + \Gamma \quad e : \tau_B}{\Gamma \quad (\lambda (x : \tau_A) \ e) : \tau_B} \quad \text{MLFUN} \]

\[ \frac{\Gamma \quad e_1 : \tau_1 \rightarrow \tau_2 \quad \Gamma \quad e_2 : \tau_1}{\Gamma \quad (e_1 \ e_2) : \tau_2} \quad \text{MLAPP} \]

\[ \frac{\Gamma \quad e_1 : \text{int} \quad \Gamma \quad e_2 : \text{int}}{\Gamma \quad (+ e_1 \ e_2) : \text{int}} \quad \text{MLSUM} \]

ML type system
\[
\begin{align*}
n & \in \{0, 1, 2, \ldots\} \\
\Gamma & \quad n : \text{TST} \\
\Gamma & \quad x : \text{TST} \in \Gamma \\
\Gamma & \quad x : \text{TST} \\
[x := \text{TST}] & + \Gamma \quad e : \text{TST} \\
\Gamma & \quad (\lambda (x) e) : \text{TST} \\
\Gamma & \quad e_1 : \text{TST} \quad \Gamma \quad e_2 : \text{TST} \\
\Gamma & \quad (e_1 e_2) : \text{TST} \\
\Gamma & \quad e_1 : \text{TST} \quad \Gamma \quad e_2 : \text{TST} \\
\Gamma & \quad (+ e_1 e_2) : \text{TST} 
\end{align*}
\]

Scheme type system
The big question:
How do we put them together?
The big question:
How do we put them together?
The big question:
How do we put them together?

Method 1: lump embedding
Anatomy of a Boundary

$(\text{MS } \tau \text{ TST } e)$

An ML boundary
Anatomy of a Boundary

\((\text{MS} \ \tau \ \text{TST} \ \varepsilon)\)

"ML outside, Scheme inside"
Anatomy of a Boundary

\((\text{MS } \tau \text{ TST } e)\)

The Scheme expression to run
Anatomy of a Boundary

\[(MS \uparrow \tau \ TST \ e)\]

The ML side's type
Anatomy of a Boundary

\((\text{MS} \; \tau \; \text{TST} \; e)\)

The Scheme side's type
Anatomy of a Boundary

... which isn't necessary to write down

\[(\text{MS } \tau \ e)\]
Anatomy of a Boundary

(\(SM \ TST \ \tau \ e\))

A Scheme boundary
Anatomy of a Boundary

The Scheme side's type

\[(\text{SM} \quad \text{TST} \quad \tau \quad e)\]
Anatomy of a Boundary

\((SM \ \tau \ e)\)

... which isn't necessary to write down
Anatomy of a Boundary

( SM \tau e )

The ML side's type
\[ e ::= v \mid (e\ e) \mid (+\ e\ e) \mid (SM\ \tau\ e) \]

\[ v ::= (\lambda\ (x)\ e) \mid \text{number} \]

New Scheme grammar
\[ e := v \mid (e \; e) \mid (+ \; e \; e) \mid (MS \; \tau \; e) \]

\[ v := (\lambda \; (x : \tau) \; e) \mid \text{number} \]

\[ \tau := \text{int} \mid (\tau \rightarrow \tau) \mid \text{TST} \]

New ML grammar
(+ (MS int 3) 4)
'shouldbe error
(+ (MS int 3) 4)
'shouldbe error

(+ (MS int (SM int 3)) 4)
'shouldbe 7
(+ (MS int 3) 4)
'shouldbe error

(+ (MS int (SM int 3)) 4)
'shouldbe 7

(+ (MS int ((λ (x) x) (SM int 3))) 4)
'shouldbe 7
(+ (MS int 3) 4)
'shouldbe error

(+ (MS int (SM int 3)) 4)
'shouldbe 7

(+ (MS int ((λ (x) x) (SM int 3))) 4)
'shouldbe 7

(+ (MS int ((λ (x) x) (SM (int → int) F))) 4)
'shouldbe error
(+ (MS int 3) 4)  
'shouldbe error

(+ (MS int (SM int 3)) 4)  
'shouldbe 7

(+ (MS int ((λ (x) x) (SM int 3))) 4)  
'shouldbe 7

(+ (MS int ((λ (x) x) (SM (int → int) F))) 4)  
'shouldbe error

(+ (MS int ((λ (x) x) (SM int (λ (x : int) x)))) 4)  
'shouldbe Type error! 
\[
C := [ ] \mid (v \ C) \mid (C \ e) \mid (MS \ \tau \ C)
\]

\[
C[((\lambda (x : \tau) \ e) \ v)] \rightarrow C[e \ [x := v]]
\]
\[
C[(+ n_1 n_2)] \rightarrow C[n_1 + n_2]
\]

New ML reductions
\[ C := \template{[ ]} \mid \template{(v \ C)} \mid \template{(C \ e)} \mid \template{(SM \ \tau \ C)} \]

\[ C[((\lambda (x) e) v)] \rightarrow C[e \ [x := v]] \]

\[ C[(+ n_1 n_2)] \rightarrow C[n_1+n_2] \]

New Scheme reductions
How do we know what reduction to use?

\[ C[(+ \ 1 \ 2)] \]

\[ C[(+ \ n_1 \ n_2)] \rightarrow C[n_1+n_2] \]

\[ C[(+ \ n_1 \ n_2)] \rightarrow C[n_1+n_2] \]
How do we know what reduction to use?

\[ C[(+ \, 1 \, 2)] \]

\[
C[( + \, n_1 \, n_2 )] \rightarrow C[n_1 + n_2]
\]

\[
C[( + \, n_1 \, n_2 )] \rightarrow C[n_1 + n_2]
\]
How do we know what reduction to use?

\[
C[+(12)]
\]

\[
C := [ ] | (v C) | (C e) | (SM \tau C)
\]

\[
C := [ ] | (v C) | (C e) | (MS \tau C)
\]

\[
C[+(n_1 n_2)] \rightarrow C[n_1+n_2]
\]

\[
C[+(n_1 n_2)] \rightarrow C[n_1+n_2]
\]
\[ n \in \{0, 1, 2, \ldots\} \]
\[ \Gamma \quad n : \text{int} \quad \text{MLNUM} \]
\[ x : \tau \in \Gamma \]
\[ \Gamma \quad x : \tau \quad \text{MLVAR} \]
\[ [x := \tau_A] + \Gamma \quad e : \tau_B \quad \text{MLFUN} \]
\[ \Gamma \quad (\lambda (x : \tau_A) \ e) : \tau_B \]
\[ \Gamma \quad e_1 : \tau_1 \rightarrow \tau_2 \quad \Gamma \quad e_2 : \tau_1 \quad \text{MLAPP} \]
\[ \Gamma \quad (e_1 \ e_2) : \tau_2 \]
\[ \Gamma \quad e_1 : \text{int} \quad \Gamma \quad e_2 : \text{int} \quad \text{MLSUM} \]
\[ \Gamma \quad (+ e_1 \ e_2) : \text{int} \]
\[ \Gamma \quad e : \text{TST} \quad \text{MLBOUNDARY} \]
\[ \Gamma \quad (\text{MS} \ \tau \ e) : \tau \]

New ML type system
\[
\frac{n \in \{0, 1, 2, \ldots\}}{\Gamma \; n : \text{TST}} \quad \text{SNUM}
\]

\[
\frac{x : \text{TST} \in \Gamma}{\Gamma \; x : \text{TST}} \quad \text{SVAR}
\]

\[
\frac{[x := \text{TST}] + \Gamma \; e : \text{TST}}{\Gamma \; (\lambda (x) e) : \text{TST}} \quad \text{SFUN}
\]

\[
\frac{\Gamma \; e_1 : \text{TST} \quad \Gamma \; e_2 : \text{TST}}{\Gamma \; (e_1 \; e_2) : \text{TST}} \quad \text{SAPP}
\]

\[
\frac{\Gamma \; e_1 : \text{TST} \quad \Gamma \; e_2 : \text{TST}}{\Gamma \; (+ e_1 \; e_2) : \text{TST}} \quad \text{SSUM}
\]

\[
\frac{\Gamma \; e : \tau}{\Gamma \; \text{(SM} \; \tau \; e \text{)} : \text{TST}} \quad \text{SBOUNDARY}
\]

New Scheme type system
Packing and unpacking values

\[ v := (\lambda (x) \; e) \mid \text{number} \mid (\text{SM} \; \tau' \; v) \]
\[ v := (\lambda (x : \; \tau) \; e) \mid \text{number} \mid (\text{MS} \; \text{TST} \; v) \]

Where \( \tau' = \tau - \{\text{TST}\} \)

\[ C[(\text{MS} \; \tau' \; (\text{SM} \; \tau' \; v))] \rightarrow C[v] \]
\[ C[(\text{MS} \; \tau' \; v)] \rightarrow \text{Error} \]
Packing and unpacking values

\[ v ::= (\lambda (x) e) \mid \text{number} \mid (\text{SM} \, \tau' \, v) \]
\[ v ::= (\lambda (x : \tau) e) \mid \text{number} \mid (\text{MS} \, \text{TST} \, v) \]

Where \( \tau' = \tau - \{\text{TST}\} \)

\[ \text{C}[(\text{MS} \, \tau' \, (\text{SM} \, \tau' \, v))] \rightarrow \text{C}[v] \]
\[ \text{C}[(\text{MS} \, \tau' \, v)] \rightarrow \text{Error} \]

\[ \text{C}[(\text{SM} \, \text{TST} \, (\text{MS} \, \text{TST} \, v))] \rightarrow \text{C}[v] \]
Packing and unpacking values

\[ v := (\lambda (x) \ e) \mid \text{number} \mid (\text{SM} \ \tau' \ v) \]
\[ v := (\lambda (x : \tau) \ e) \mid \text{number} \mid (\text{MS} \ \text{TST} \ v) \]

Where \( \tau' = \tau - \{\text{TST}\} \)

\[
\text{C}[ (\text{MS} \ \tau' \ (\text{SM} \ \tau' \ v)) ] \to \text{C}[v] \\
\text{C}[ (\text{MS} \ \tau' \ v) ] \to \text{Error} \\
\text{C}[ (\text{SM} \ \text{TST} \ (\text{MS} \ \text{TST} \ v)) ] \to \text{C}[v] \\
\text{C}[ (\text{SM} \ \text{TST} \ v) ] \to \text{Won't typecheck!} 
\]
(+ (MS int 
  ((λ (x) x) (SM int 3)))
 4)
\[ (+ \text{ MS int }\]

\[
((\lambda (x) (x \ x))
  
((\lambda (x) (x x))))
\]

\[ 4) \]
Aside (#1):
What if there were no Earth?
; H (hide) : (TST -> TST) -> TST
(define H
  (λ (a : (TST -> TST))
  (MS TST (λ (x) (SM (TST -> TST) a))))))

; U (unhide) : TST -> (TST -> TST)
(define U
  (λ (a : TST)
  (MS (TST -> TST) ((SM TST a) 1)))))

(   (λ (x : TST) ((U x) x))
 (H (λ (x : TST) ((U x) x))))
; H (hide) : (TST -> TST) -> TST
(define H
  (λ (a : (TST -> TST))
    (MS TST (λ (x) (SM (TST -> TST) a)))))

; U (unhide) : TST -> (TST -> TST)
(define U
  (λ (a : TST)
    (MS (TST -> TST) ((SM TST a) 1))))

(  (λ (x : TST) (((U x) x))
(  (H (λ (x : TST) (((U x) x))))))
Theorem:
A lump-embedding program that typechecks never goes wrong
Less formal theorem:
A lump-embedding program that typechecks never does anything useful
Back to the big question:
How else can we put them together?
Back to the big question:
How else can we put them together?

Method 2: natural embedding
Convert equals for equals

(MS int 4)
'shouldbe 4
Convert equals for equals

(MS int 4)
'should be 4

(((MS (int -> int) (λ (x) x)) 2)
'should be 2
Convert equals for equals

(MS int 4)
'shouldbe 4

((MS (int -> int) (λ (x) x)) 2)
'shouldbe 2

(MS int (λ (x) x))
'shouldbe error
Convert equals for equals

\[(\text{MS int } 4)\]
'should be 4

\[((\text{MS (int } \rightarrow \text{ int) } (\lambda (x) x)) \ 2)\]
'should be 2

\[(\text{MS int } (\lambda (x) x))\]
'should be error

\[(\text{SM int } (\lambda (x : \text{ int) } x))\]
'should be type error
How?

$$(\text{MS int 4}) \rightarrow 4$$
How?

\[(\text{MS int} \ 4) \rightarrow 4\]

\[(\text{MS (int -> int)} \ (\lambda \ (x) \ x)) \rightarrow (\lambda \ (x : \text{int}) \ x)\]
How?

\[(\text{MS int } 4) \rightarrow 4\]

\[(\text{MS (int -> (! (x) x))) } \rightarrow (\lambda (x : \text{int}) x)\]
How?

\[(\text{MS int } 4) \rightarrow 4\]

\[(\text{MS (int -> int) } (\lambda (x) x)) \rightarrow (\lambda (x : \text{int}) (\text{MS int } ((\lambda (x) x) (\text{SM int } x))))\]
How?

\[(\text{MS int } 4) \rightarrow 4\]

\[(\text{MS (int -> int) (}\lambda (x) x)) \rightarrow (\lambda (x : \text{int}) (\text{MS int ((}\lambda (x) x) (\text{SM int x})))))\]

But what if the Scheme code doesn't produce an int?
Even worse:

\[
(\lambda (x : \text{int})
  (\text{MS} (\text{int} \to \text{int})
    ((\lambda (a) (\lambda (b) (\lambda (c) c)))
     (\text{MS} \text{int} x))))
\]

What if ML can't immediately tell that something is wrong?
\[(\text{MS} \ ((\text{int} \to \text{int}) \to (\text{int} \to \text{int})) \ F)\]

\[
\rightarrow
\]

\[(\lambda \ (x : (\text{int} \to \text{int}))
\quad (\text{MS} \ (\text{int} \to \text{int})
\quad (F \ (\text{SM} \ (\text{int} \to \text{int}) \ x))))\]
(MS ((int -> int) -> (int -> int)) F)

→

(\( \lambda \) (x : (int -> int))
  (MS (int -> int)
    (G (int -> int)
      (F (SM (int -> int) x)))))

guard the context from bad Scheme values
(MS ((int -> int) -> (int -> int)) F) 

→ 

(λ (x : (int -> int))
 (MS (int -> int)
 (G (int -> int)
 (F (G (int -> int)
 (SM (int -> int) x)))))) 


guard the ML function from wrong uses
These are two different jobs
Anatomy of a Guard

$$(G^+ \tau \nu)$$

All guards are projections on values
Anatomy of a Guard

\[(G^+ \quad \tau \quad v)\]

All guards are in Scheme
Anatomy of a Guard

Positive guards (jailors): "Make v behave like a tau"
Anatomy of a Guard

\[(G^+ \quad \tau \quad v)\]

v is the value we're projecting
Anatomy of a Guard

\[(G^+ \tau v)\]

the type \(v\) must behave like
Anatomy of a Guard

\[
(G^- \tau v)
\]

Negative guards (bodyguards): "Make the context treat v like a tau"
e ::= v | (e e) | (+ e e) | (SM τ e) | (G^+ τ e) | (G^- τ e)

v ::= (λ (x) e) | number

C ::= [] | (v C) | (C e) | (SM τ C) | (G^+ τ C) | (G^- τ C)
\[ \begin{align*}
e & := v \mid (e \ e) \mid (+ \ e \ e) \mid (\text{MS } \tau \ e) \\
v & := (\lambda (x : \tau) \ e) \mid \text{number} \\
\tau & := \text{int} \mid (\tau \to \tau) \\
C & := [] \mid (v \ C) \mid (C \ e) \mid (\text{MS } \tau \ C)\end{align*} \]
\((G^+ \text{ int } n) \rightarrow n\)
if \(n\) is a number, error otherwise

\((G^+ (\tau_1 \rightarrow \tau_2) f) \rightarrow (\lambda (x) (G^+ \tau_1 ((f (G^- \tau_2 x))))))\)
if \(f\) is a procedure value, error otherwise
\[(G^+ \text{ int } n) \rightarrow n\]
if \(n\) is a number, error otherwise

\[(G^+ (\tau_1 \rightarrow \tau_2) f) \rightarrow (\lambda (x) (G^+ \tau_1 ((f (G^- \tau_2 x))))))\]
if \(f\) is a procedure value, error otherwise

\[(G^- \text{ int } n) \rightarrow n\]

\[(G^- (\tau_1 \rightarrow \tau_2) f) \rightarrow (\lambda (x) (G^- \tau_1 ((f (G^+ \tau_2 x))))))\]

No direct dynamic checks - negative guards are trusting
Aside (#2):
What if there were no Mars?
• Guards don't care about the type system (only vice versa)
• Scheme embedded in Scheme could use the same technique, and does
Back to the big question again:
How well have we answered it?
• A method for modeling multilanguage semantics
• Boundaries, recursive contexts
• Simple but models a lot
• This is an interesting way to talk about language interoperation!
The End