A Semantic Approach to Multi-Language Systems

Jacob Matthews
University of Chicago
jacobm@cs.uchicago.edu
A Semantic Approach to Multi-Language Systems

Jacob Matthews
University of Chicago
jacobm@cs.uchicago.edu
A Semantic Approach to Multi-Language Systems

Jacob Matthews
University of Chicago - Irvine Campus
jacobm@cs.uchicago.edu
ML grammar

```plaintext
e ::= v | (e e) | (+ e e)

v ::= (λ (x : τ) e) | number

τ ::= int | (τ → τ)

E ::= [] | (v E) | (E e) | (+ E e) | (+ v E)

ML grammar
```
\[ E[(\lambda (x : \tau) \ e) \ v)] \quad E[e \ [x := v]] \]

\[ E[(+ \ n_1 \ n_2)] \quad E[n_1+n_2] \]

ML reductions
\begin{align*}
e & := v \mid (e \ e) \mid (+ \ e \ e) \\
v & := (\lambda \ (x) \ e) \mid \text{number} \\
E & := [] \mid (v \ E) \mid (E \ e) \mid (+ \ E \ e) \mid (+ \ v \ E)
\end{align*}

Scheme grammar
\[E[(\lambda (x) \ e) \ v)]\]  \[E[e \ [x := v]]\]  
\[E[(+ n_1 n_2)]\]  \[E[n_1+n_2]\]  
\[E[(n \ v)]\]  \textbf{Error}: can't apply numbers  
\[E[(+ (\lambda (x) \ e) \ n)]\]  \textbf{Error}: can't add functions  
\[E[(+ n (\lambda (x) \ e))]\]  \textbf{Error}: can't add functions  

\textbf{Scheme reductions}
The key idea: boundaries

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

An ML boundary
The key idea: boundaries

(MS τ TST e)

"ML outside, Scheme inside"
The key idea: boundaries

The Scheme expression to run

\((\text{MS} \ \tau \ \text{TST} \ e)\)
The key idea: boundaries

\((MS \quad \tau \quad TST \quad e)\)

The ML side's type
The key idea: boundaries

\((MS \; \tau \; TST \; e)\)

The Scheme side's type
The key idea: boundaries

\[(MS \quad \tau \quad e)\]

... which isn't necessary to write down
The key idea: boundaries

\[(SM \ TST \ \tau \ e)\]

A Scheme boundary
The key idea: boundaries

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

The Scheme side's type
The key idea: boundaries

... which isn't necessary to write down
The key idea: boundaries

\((SM \tau e)\)

The ML side's type
New ML grammar

e ::= v | (e e) | (+ e e) | (MS τ e)

v ::= (λ (x : τ) e) | number

τ ::= int | (τ → τ)
New Scheme grammar

\[ e ::= v \mid (e \ e) \mid (+ \ e \ e) \mid (SM \ \tau \ e) \]
\[ v ::= (\lambda \ (x) \ e) \mid \text{number} \]
E ::= [] | (v E) | (E e) | (MS τ E)

E[((λ (x : τ) e) v)] E[e [x := v]]
E[+] E[n1+n2]

New ML reductions
\[ E[(\text{MS int } n)] \quad E[n] \]

\[ E[(\text{MS } \tau_1 \rightarrow \tau_2 \ (\lambda \ (x) \ e))] \]
\[ E[(\lambda \ (x : \tau_1) \ (\text{MS } \tau_2 \ ((\lambda \ (x) \ e) \ (\text{SM } \tau_1 \ x))))] \]

\[ E[(\text{MS int } (\lambda \ (x) \ e))] \quad \text{Error: Non-number} \]
\[ E[(\text{MS } \tau_1 \rightarrow \tau_2 \ n)] \quad \text{Error: Non-function} \]
((\text{MS} \ (\text{int} \to \text{int}) \ (\lambda \ (x) \ (+ \ x \ 1))) \ 2) \\
((\lambda \ (y : \text{int}) \\
\quad (\text{MS} \ \text{int} \ ((\lambda \ (x) \ (+ \ x \ 1)) \ (\text{SM} \ \text{int} \ y)))) \\
\quad 2) \\
(\text{MS} \ \text{int} \ ((\lambda \ (x) \ (+ \ x \ 1)) \ (\text{SM} \ \text{int} \ 2))) \\
(\text{MS} \ \text{int} \ ((\lambda \ (x) \ (+ \ x \ 1)) \ 2)) \\
(\text{MS} \ \text{int} \ (+ \ 2 \ 1)) \\
(\text{MS} \ \text{int} \ 3) \\
3
(((MS (int → int) (λ (x) (+ x 1))) 2))

(((λ (y : int)
    (MS int ((λ (x) (+ x 1)) (SM int y)))) 2))

(MS int ((λ (x) (+ x 1)) (SM int 2)))

(MS int ((λ (x) (+ x 1)) 2))

(MS int (+ 2 1))

(MS int 3)

3
((MS (int → int) (λ (x) (+ x 1))) 2)  
((λ (y : int)  
    (MS int ((λ (x) (+ x 1)) (SM int y)))) 2)  
(MS int ((λ (x) (+ x 1)) (SM int 2)))  
(MS int ((λ (x) (+ x 1)) 2))  
(MS int (+ 2 1))  
(MS int 3)
3
(((\text{MS} \int \rightarrow \int) \ (\lambda \ (x) \ (+ \ x \ 1))) \ 2)

(((\lambda \ (y : \int)
   \ (\text{MS} \int \ ((\lambda \ (x) \ (+ \ x \ 1)) \ (\text{SM} \int \ y)))))) \ 2)

(\text{MS} \int \ ((\lambda \ (x) \ (+ \ x \ 1)) \ (\text{SM} \int \ 2)))

(\text{MS} \int \ ((\lambda \ (x) \ (+ \ x \ 1)) \ 2))

(\text{MS} \int \ (+ \ 2 \ 1))

(\text{MS} \int \ 3)

3
((MS (int -> int) (λ (x) (+ x 1))) 2)

((λ (y : int)
    (MS int ((λ (x) (+ x 1)) (SM int y))))
  2)

(MS int ((λ (x) (+ x 1)) (SM int 2)))

(MS int ((λ (x) (+ x 1)) 2))

(MS int (+ 2 1))

(MS int 3)

3
((MS (int → int) (λ (x) (+ x 1))) 2)
((λ (y : int)
    (MS int ((λ (x) (+ x 1)) (SM int y))))
  2)
(MS int ((λ (x) (+ x 1)) (SM int 2)))
(MS int ((λ (x) (+ x 1)) 2))
(MS int (+ 2 1))
(MS int 3)
((MS (int → int) (λ (x) (+ x 1))) 2)

((λ (y : int)
   (MS int ((λ (x) (+ x 1)) (SM int y)))) 2)

(MS int ((λ (x) (+ x 1)) (SM int 2)))

(MS int ((λ (x) (+ x 1)) 2))

(MS int (+ 2 1))

(MS int 3)

3
{(MS (int → int) (λ (x) (λ (y) y))) 2)

((λ (y : int)
   (MS int ((λ (x) (λ (y) y)) (SM int y))))
  2)

(MS int ((λ (x) (λ (y) y)) (SM int 2)))

(MS int ((λ (x) (λ (y) y)) 2))

(MS int (λ (y) y))

Error: Non-number
((MS \ (\text{int} \to \text{int}) \ (\lambda \ (x) \ (\lambda \ (y) \ y))) \ 2) \\
((\lambda \ (y : \text{int}) \\
\quad \text{(MS int } ((\lambda \ (x) \ (\lambda \ (y) \ y)) \ (\text{SM int y})))))) \\
2) \\
\text{(MS int } ((\lambda \ (x) \ (\lambda \ (y) \ y)) \ (\text{SM int 2}))) \\
\text{(MS int } ((\lambda \ (x) \ (\lambda \ (y) \ y)) \ 2)) \\
\text{(MS int } (\lambda \ (y) \ y)) \\
\text{Error: Non-number}
(((MS (int → int) (λ (x) (λ (y) y))) 2)
((λ (y : int)
  (MS int (((λ (x) (λ (y) y)) (SM int y)))))) 2)
(MS int (((λ (x) (λ (y) y)) (SM int 2))))
(MS int (((λ (x) (λ (y) y)) 2))
(MS int (λ (y) y))

Error: Non-number
((MS (int → int) (λ (x) (λ (y) y))) 2)

((λ (y : int)
   (MS int ((λ (x) (λ (y) y)) (SM int y))))
  2)

(MS int ((λ (x) (λ (y) y)) (SM int 2)))

(MS int ((λ (x) (λ (y) y)) 2))

(MS int (λ (y) y))

Error: Non-number
\[ ((\text{MS} \ (\text{int} \to \text{int})) \ (\lambda \ (x) \ (\lambda \ (y) \ y))) \ 2) \]

\[ ((\lambda \ (y : \text{int}) \ ((\text{MS} \ \text{int} \ ((\lambda \ (x) \ (\lambda \ (y) \ y)) \ (\text{SM} \ \text{int} \ y)))) \ 2) \]

\[ (\text{MS} \ \text{int} \ ((\lambda \ (x) \ (\lambda \ (y) \ y)) \ (\text{SM} \ \text{int} \ 2))) \]

\[ (\text{MS} \ \text{int} \ ((\lambda \ (x) \ (\lambda \ (y) \ y)) \ 2)) \]

\[ (\text{MS} \ \text{int} \ (\lambda \ (y) \ y)) \]

\textbf{Error: Non-number}
(((MS (int → int) (λ (x) (λ (y) y)))) 2)

(((λ (y : int)
  (MS int (((λ (x) (λ (y) y)) (SM int y))))
   2)

(MS int (((λ (x) (λ (y) y)) (SM int 2))))

(MS int (((λ (x) (λ (y) y)) 2))

(MS int (λ (y) y))

Error: Non-number
**Theorem** (type soundness):

If $e : \tau$, then either: $e \ast v$;  
    or $e \ast \textbf{Error: } \text{msg}$;  
    or $e$ diverges.

**Proof:** Mutually-recursive variant of standard preservation and progress.
Information hiding = parametric polymorphism
[Reynolds]

versus

Information hiding = generativity + scope
[Morris]
Information hiding = parametric polymorphism

∀α.α ⊆ α
Information hiding = generativity + scope

;; -key
(define (create-key) (gensym))

;; TST key -key \rightarrow TST
(define (lock v key1)
  (\ key2
    (if (eq? key1 key2)
      v
      (error)))))
Question:
But do these two strategies produce the same result?
**Question:**
But do these two strategies produce the same result?

*Actually, that's a bad question.*
Better question:
Can we use dynamic seals to combine polymorphic ML and Scheme without breaking parametricity?
Add generic types to ML:

\[
\begin{align*}
e & := v \mid (e \; e) \mid (+ \; e \; e) \mid (\text{MS} \; \tau \; e) \mid e^{\langle \tau \rangle} \\
v & := (\lambda \; (x : \tau) \; e) \mid \text{number} \mid (\forall \alpha. \; e) \\
\tau & := \text{int} \mid (\tau \rightarrow \tau) \mid \alpha \mid \forall \alpha. \; \tau
\end{align*}
\]
What about the new boundaries?

\((SM \ \alpha \ e)\)

\((MS \ \alpha \ e)\)

\((SM \ \forall \alpha. \tau \ e)\)

\((MS \ \forall \alpha. \tau \ e)\)
What about the new boundaries?
What about the new boundaries?

$$(\text{SM} \: \alpha \: e)$$

$$(\text{MS} \: \alpha \: e)$$
What about the new boundaries?

\[(\text{SM} \ \alpha \ \nu)\]

\[(\text{MS} \ \alpha \ \nu)\]
What about the new boundaries?

Typed value entering untyped code: Seal!
What about the new boundaries?

\[
\left( \text{SM} \ \alpha \ \nu \right) \quad \left( \text{MS} \ \alpha \ \nu \right)
\]

Untyped value claimed to satisfy type $\alpha$: Unseal!
Theorem (parametricity):
  - The ML side of the embedding is parametric
  - The Scheme side of the embedding is oblivious to the contents of sealed values

Proof: Simultaneously show both claims using the method of logical relations. (We need step-indexed logical relations to handle Scheme.)
Consequences:

• Higher-order polymorphic contracts with dynamic sealing enforce parametricity

• At least some of System F’s good properties are preserved by a well-designed embedding
Consequences:

- Higher-order polymorphic contracts with dynamic sealing enforce parametricity

- At least some of System F’s good properties are preserved by a well-designed embedding

... But what about all of them?
A design criterion for multi-language interfaces
\[ a = b \]
\[ a = b \]
\[ a = b \]
\[ a = b \]
\[a = b\]

\[a = b\]

\[a \neq b\]
\[ a = b \]

\[ a = b \]

\[ a = b \]

\[ a \neq b \]

This is strange ...
Proposed multi-language design criterion:

For any two expressions $a$ and $b$,

\[ a \cong^P b \]

implies

\[ a \cong^{P+Q} b \]
Thanks!