

Full abstraction for multi-language systems

ML plus linear types

Gabriel Scherer, Amal Ahmed, Max New

Northeastern University, Boston

January 15, 2017

Multi-language systems

Languages of today tend to evolve into behemoths by piling features up: C++, Scala, GHC Haskell, OCaml...

Multi-language systems: several languages working together to cover the feature space. (simpler?)

Multi-language system **design** may include designing new languages for interoperation.

Full abstraction to understand graceful language interoperability.

Full abstraction for multi-language systems

$\llbracket _ \rrbracket : S \rightarrow T$ fully abstract:

$$a \approx^{ctx} b \implies \llbracket a \rrbracket \approx^{ctx} \llbracket b \rrbracket$$

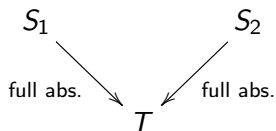
Full abstraction preserves (equational) reasoning.

Full abstraction for multi-language systems

$\llbracket _ \rrbracket : S \rightarrow T$ fully abstract:

$$a \approx^{ctx} b \implies \llbracket a \rrbracket \approx^{ctx} \llbracket b \rrbracket$$

Full abstraction preserves (equational) reasoning.



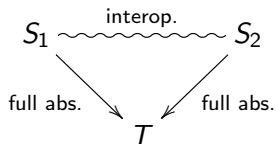
Mixed S_1, S_2 programs preserve (equational) reasoning of their fragments.

Full abstraction for multi-language systems

$\llbracket _ \rrbracket : S \rightarrow T$ fully abstract:

$$a \approx^{ctx} b \implies \llbracket a \rrbracket \approx^{ctx} \llbracket b \rrbracket$$

Full abstraction preserves (equational) reasoning.



Mixed S_1, S_2 programs preserve (equational) reasoning of their fragments.
Graceful multi-language semantics.

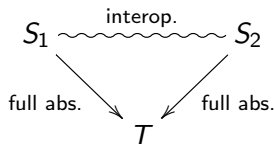
(or vice versa)

Full abstraction for multi-language systems

$\llbracket _ \rrbracket : S \rightarrow T$ fully abstract:

$$a \approx^{ctx} b \implies \llbracket a \rrbracket \approx^{ctx} \llbracket b \rrbracket$$

Full abstraction preserves (equational) reasoning.



Mixed S_1, S_2 programs preserve (equational) reasoning of their fragments.
Graceful multi-language semantics.

(or vice versa)

In this talk: a first ongoing experiment on **ML** plus **linear types**.

U: a core ML

$$\Gamma \vdash_{\mathbf{u}} e : \sigma$$

L: linear types

Resource tracking, unique ownership.

$$\begin{array}{cccc} \sigma & !\sigma & \Gamma & !\Gamma \\ \Gamma \vdash_l e : \sigma & & & \end{array}$$

We own e at type σ (duplicable or not), e owns the resources in Γ .

Multi-language applications

Protocol with resource handling requirements.

“This file descriptor must be closed”

open : $!(\![\text{Path}] \multimap \text{Handle})$
line : $!(\text{Handle} \multimap (\text{Handle} \oplus (\![\text{String}] \otimes \text{Handle})))$
close : $!(\text{Handle} \multimap \mathbf{1})$

(details about the boundaries come later)

Typestate.

(details about the boundaries come later)

```
open  : !( ![Path]  $\multimap$  Handle )
line  : ! ( Handle  $\multimap$  ( Handle  $\oplus$  ( ![String]  $\otimes$  Handle )))
close : ! ( Handle  $\multimap$  1 )
```

```
let concat_lines path : String = UL(
  loop (open LU(path)) LU(Nil)
  where rec loop handle (acc : ![List String]) =
    match line handle with
    | EOF handle ->
      close handle; LU(rev_concat "\n" UL(acc))
    | Next line handle ->
      loop handle LU(Cons UL(line) UL(acc)))
```

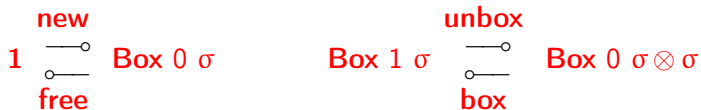
$$\frac{!\Gamma \vdash_{lu} e : \sigma}{!\Gamma \vdash_{ul} \mathcal{LU}(e) : ![\sigma]}$$

$$\frac{!\Gamma \vdash_{ul} e : ![\sigma]}{!\Gamma \vdash_{lu} \mathcal{UL}(e) : \sigma}$$

Linear types: linear locations

Box 1 σ : full cell

Box 0 σ : empty cell



Applications: in-place reuse of memory cells.

List reversal

```
type LList a =  $\mu t. 1 \oplus \text{Box } 1 (a \otimes t)$   
pattern Nil = inl ()  
pattern Cons l x xs = inr (box (l, (x, xs)))
```

```
val reverse : LList a  $\multimap$  LList a  
let reverse list = loop Nil list  
  where rec loop tail = function  
    | Nil  $\rightarrow$  tail  
    | Cons l x xs  $\rightarrow$  loop (Conx l x tail) xs
```

```
type List a =  $\mu t. 1 + (a \times t)$   
let reverse list = UL(share (reverse (copy (LU(list)))))
```

$$\vdash_{ul} \sigma \simeq \sigma$$

Full abstraction

Theorem

The embedding of \mathbf{U} into \mathbf{UL} is fully abstract.

Proof: by pure interpretation of the linear language into ML.
(Cogent)

Questions ?

Thanks!

Interaction: lump

Types $\sigma \mid \sigma$

σ

$\sigma \quad + ::= \dots \mid [\sigma]$

Values $v \mid v$

v

$v \quad + ::= \dots \mid [v]$

Expressions $e \mid e$

$e \quad + ::= \dots \mid \mathcal{UL}(e)$

$e \quad + ::= \dots \mid \mathcal{LU}(e)$

Contexts $\Gamma ::= \cdot \mid \Gamma, x:\sigma \mid \Gamma, \alpha \mid \Gamma, \mathbf{x}:\sigma$

$$\frac{!\Gamma \vdash_{\text{lu}} e : \sigma}{!\Gamma \vdash_{\text{ul}} \mathcal{LU}(e) : ![\sigma]}$$

$$\frac{!\Gamma \vdash_{\text{ul}} e : ![\sigma]}{!\Gamma \vdash_{\text{lu}} \mathcal{UL}(e) : \sigma}$$

Interaction: compatibility

Compatibility relation $\boxed{\vdash_{\text{ul}} \sigma \simeq \sigma}$

$$\frac{}{\vdash_{\text{ul}} \mathbf{1} \simeq !\mathbf{1}} \qquad \frac{\vdash_{\text{ul}} \sigma_1 \simeq !\sigma_1 \quad \vdash_{\text{ul}} \sigma_2 \simeq !\sigma_2}{\vdash_{\text{ul}} \sigma_1 \times \sigma_2 \simeq !(\sigma_1 \otimes \sigma_2)}$$

$$\frac{\vdash_{\text{ul}} \sigma_1 \simeq !\sigma_1 \quad \vdash_{\text{ul}} \sigma_2 \simeq !\sigma_2}{\vdash_{\text{ul}} \sigma_1 + \sigma_2 \simeq !(\sigma_1 \oplus \sigma_2)} \qquad \frac{\vdash_{\text{ul}} \sigma \simeq !\sigma \quad \vdash_{\text{ul}} \sigma' \simeq !\sigma'}{\vdash_{\text{ul}} \sigma \rightarrow \sigma' \simeq !(\sigma \multimap \sigma')}$$

$$\frac{}{\vdash_{\text{ul}} \sigma \simeq ![\sigma]} \qquad \frac{\vdash_{\text{ul}} \sigma \simeq !\sigma}{\vdash_{\text{ul}} \sigma \simeq !!\sigma} \qquad \frac{\vdash_{\text{ul}} \sigma \simeq !\sigma}{\vdash_{\text{ul}} \sigma \simeq !(\mathbf{Box} \ 1 \ \sigma)}$$

Interaction primitives and derived constructs:

$$!\sigma \quad \begin{array}{c} \overset{\sigma \text{ unlump}}{\text{---} \circ} \\ \text{---} \circ \\ \underset{\text{lump}^\sigma}{\text{---}} \end{array} \quad \sigma \quad \text{when} \quad \vdash_{\text{ul}} \sigma \simeq \sigma \qquad \begin{array}{l} \sigma \mathcal{LU}(e) \stackrel{\text{def}}{=} \sigma \text{ unlump } \mathcal{LU}(e) \\ \mathcal{UL}^\sigma(e) \stackrel{\text{def}}{=} \mathcal{UL}(\text{lump}^\sigma e) \end{array}$$