

Full abstraction for multi-language systems

ML plus linear types

Gabriel Scherer, Amal Ahmed, Max New

Northeastern University

August 18, 2016

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:

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

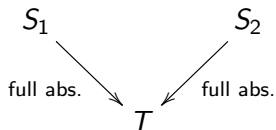
Full abstraction preserves (equational) reasoning.

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.



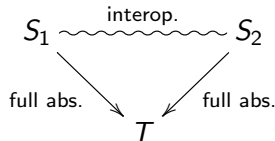
Mixed S_1, S_2 programs preserve (equational) reasoning of their fragments.

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)

Which languages?

ML sweet spot hard to beat,
but ML programmers yearn for language extensions.

ML plus:

- low-level memory, resource tracking, ownership
- effect system
- theorem proving
- ...

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

Linear types: base

A **simple** but **useful** language with linear types.

$$\frac{\Gamma_1 \vdash \sigma_1 \quad \Gamma_2 \vdash \sigma_2}{\Gamma_1 \Downarrow \Gamma_2 \vdash \sigma_1 \otimes \sigma_2}$$

$$\frac{\Gamma \vdash \sigma_1 \otimes \sigma_2 \quad \Delta, \sigma_1, \sigma_2 \vdash \sigma}{\Gamma \Downarrow \Delta \vdash \sigma}$$

$$\frac{}{!\Gamma \vdash 1}$$

$$\frac{\Gamma \vdash 1 \quad \Delta \vdash \sigma}{\Gamma \Downarrow \Delta \vdash \sigma}$$

$$\frac{\Gamma, \sigma \vdash \sigma'}{\Gamma \vdash \sigma \multimap \sigma'}$$

$$\frac{\Gamma \vdash \sigma' \multimap \sigma \quad \Delta \vdash \sigma'}{\Gamma \Downarrow \Delta \vdash \sigma}$$

$$\frac{!\Gamma \vdash \sigma}{!\Gamma \vdash !\sigma}$$

$$\frac{\Gamma \vdash !\sigma}{\Gamma \vdash \sigma}$$

Linear types: base

A **simple** but **useful** language with linear types.

$$\frac{\Gamma_1 \vdash e_1 : \sigma_1 \quad \Gamma_2 \vdash e_2 : \sigma_2}{\Gamma_1 \Downarrow \Gamma_2 \vdash \langle e_1, e_2 \rangle : \sigma_1 \otimes \sigma_2}$$

$$\frac{\Gamma \vdash e : \sigma_1 \otimes \sigma_2 \quad \Delta, x_1 : \sigma_1, x_2 : \sigma_2 \vdash e' : \sigma}{\Gamma \Downarrow \Delta \vdash \text{let } \langle x_1, x_2 \rangle = e \text{ in } e' : \sigma}$$

$$\frac{}{!\Gamma \vdash \langle \rangle : \mathbf{1}}$$

$$\frac{\Gamma \vdash e : \mathbf{1} \quad \Delta \vdash e' : \sigma}{\Gamma \Downarrow \Delta \vdash e; e' : \sigma}$$

$$\frac{\Gamma, x : \sigma \vdash e : \sigma'}{\Gamma \vdash \lambda(x : \sigma). e : \sigma \multimap \sigma'}$$

$$\frac{\Gamma \vdash e : \sigma' \multimap \sigma \quad \Delta \vdash e' : \sigma'}{\Gamma \Downarrow \Delta \vdash e e' : \sigma}$$

$$\frac{!\Gamma \vdash e : \sigma}{!\Gamma \vdash \quad :!\sigma}$$

$$\frac{\Gamma \vdash e : !\sigma}{\Gamma \vdash \quad : \sigma}$$

Linear types: base

A **simple** but **useful** language with linear types.

$$\frac{\Gamma_1 \vdash e_1 : \sigma_1 \quad \Gamma_2 \vdash e_2 : \sigma_2}{\Gamma_1 \Downarrow \Gamma_2 \vdash \langle e_1, e_2 \rangle : \sigma_1 \otimes \sigma_2}$$

$$\frac{\Gamma \vdash e : \sigma_1 \otimes \sigma_2 \quad \Delta, x_1 : \sigma_1, x_2 : \sigma_2 \vdash e' : \sigma}{\Gamma \Downarrow \Delta \vdash \text{let } \langle x_1, x_2 \rangle = e \text{ in } e' : \sigma}$$

$$\frac{}{!\Gamma \vdash \langle \rangle : 1}$$

$$\frac{\Gamma \vdash e : 1 \quad \Delta \vdash e' : \sigma}{\Gamma \Downarrow \Delta \vdash e; e' : \sigma}$$

$$\frac{\Gamma, x : \sigma \vdash e : \sigma'}{\Gamma \vdash \lambda(x : \sigma). e : \sigma \multimap \sigma'}$$

$$\frac{\Gamma \vdash e : \sigma' \multimap \sigma \quad \Delta \vdash e' : \sigma'}{\Gamma \Downarrow \Delta \vdash e e' : \sigma}$$

$$\frac{!\Gamma \vdash e : \sigma}{!\Gamma \vdash \text{share}^\sigma e : !\sigma}$$

$$\frac{\Gamma \vdash e : !\sigma}{\Gamma \vdash \quad : \sigma}$$

Linear types: base

A **simple** but **useful** language with linear types.

$$\frac{\Gamma_1 \vdash e_1 : \sigma_1 \quad \Gamma_2 \vdash e_2 : \sigma_2}{\Gamma_1 \Downarrow \Gamma_2 \vdash \langle e_1, e_2 \rangle : \sigma_1 \otimes \sigma_2}$$

$$\frac{\Gamma \vdash e : \sigma_1 \otimes \sigma_2 \quad \Delta, x_1 : \sigma_1, x_2 : \sigma_2 \vdash e' : \sigma}{\Gamma \Downarrow \Delta \vdash \text{let } \langle x_1, x_2 \rangle = e \text{ in } e' : \sigma}$$

$$\frac{}{!\Gamma \vdash \langle \rangle : 1}$$

$$\frac{\Gamma \vdash e : 1 \quad \Delta \vdash e' : \sigma}{\Gamma \Downarrow \Delta \vdash e; e' : \sigma}$$

$$\frac{\Gamma, x : \sigma \vdash e : \sigma'}{\Gamma \vdash \lambda(x : \sigma). e : \sigma \multimap \sigma'}$$

$$\frac{\Gamma \vdash e : \sigma' \multimap \sigma \quad \Delta \vdash e' : \sigma'}{\Gamma \Downarrow \Delta \vdash e e' : \sigma}$$

$$\frac{!\Gamma \vdash e : \sigma}{!\Gamma \vdash \text{share}^\sigma e : !\sigma}$$

$$\frac{\Gamma \vdash e : !\sigma}{\Gamma \vdash \text{copy}^\sigma e : \sigma}$$

Applications

Protocol with resource handling requirements.

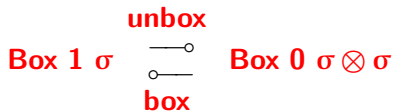
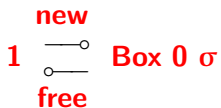
“This file descriptor must be closed”

Typestate.

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
```

(* use reverse internally *)

(* on the ML side *)

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

Full abstraction

The ML language can be compiled into a linear language.

Full abstraction

The ML language can be compiled into a linear language.

$$\begin{array}{l} \mathcal{LU}(\sigma) \stackrel{\text{def}}{=} ![\sigma] \\ \quad [\sigma_1 \times \sigma_2] \stackrel{\text{def}}{=} \mathbf{Box\ 1}([\sigma_1] \otimes [\sigma_2]) \\ \quad [1] \stackrel{\text{def}}{=} \mathbf{1} \\ \quad [\sigma_1 \rightarrow \sigma_2] \stackrel{\text{def}}{=} \mathcal{LU}(\sigma_1) \multimap \mathcal{LU}(\sigma_2) \end{array}$$

This gives a direct multi-language semantics.

Full abstraction

The ML language can be compiled into a linear language.

$$\begin{array}{l}
 \mathcal{LU}(\sigma) \stackrel{\text{def}}{=} \mathbf{!}[\sigma] \\
 \\
 [\sigma_1 \times \sigma_2] \stackrel{\text{def}}{=} \mathbf{Box\ 1} ([\sigma_1] \otimes [\sigma_2]) \\
 [1] \stackrel{\text{def}}{=} \mathbf{1} \\
 [\sigma_1 \rightarrow \sigma_2] \stackrel{\text{def}}{=} \mathcal{LU}(\sigma_1) \multimap \mathcal{LU}(\sigma_2)
 \end{array}$$

This gives a direct multi-language semantics.

Full abstraction by pure interpretation of the linear language in ML.

Full abstraction

The ML language can be compiled into a linear language.

$$\begin{array}{lcl} \mathcal{LU}(\sigma) & \stackrel{\text{def}}{=} & ![\sigma] \\ & & [\sigma_1 \times \sigma_2] \stackrel{\text{def}}{=} \mathbf{Box\ 1} ([\sigma_1] \otimes [\sigma_2]) \\ & & [1] \stackrel{\text{def}}{=} \mathbf{1} \\ & & [\sigma_1 \rightarrow \sigma_2] \stackrel{\text{def}}{=} \mathcal{LU}(\sigma_1) \multimap \mathcal{LU}(\sigma_2) \end{array}$$

This gives a direct multi-language semantics.

Full abstraction by pure interpretation of the linear language in ML.

$$\begin{array}{lcl} [!\sigma] & \stackrel{\text{def}}{=} & [\sigma] \\ [\mathbf{Box\ 1}\ \sigma] & \stackrel{\text{def}}{=} & [\sigma] \\ [\mathbf{Box\ 0}\ \sigma] & \stackrel{\text{def}}{=} & \mathbf{1} \\ [\sigma_1 \otimes \sigma_2] & \stackrel{\text{def}}{=} & [\sigma_1] \times [\sigma_2] \end{array}$$

(Cogent)

Going further

Polymorphism not formalized yet.

Implementation?