Full abstraction for multi-language systems
ML plus linear types

Gabriel Scherer, Amal Ahmed, Max New

Northeastern University, Boston

February 9, 2017

Section 1

Full Abstraction for Multi-Language Systems: Introduction

Multi-language systems

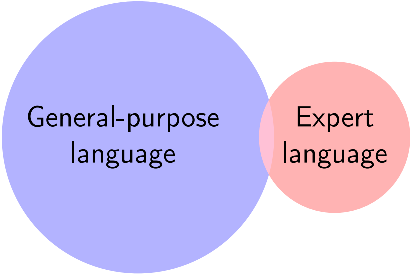
Languages of today tend to evolve into behemoths by piling features up: C++, Scala, GHC Haskell, OCaml...

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

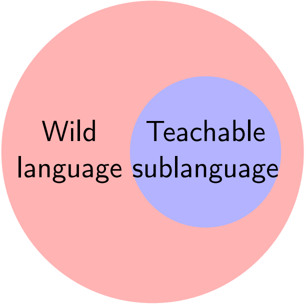
Multi-language system **design** may include designing new languages for interoperation.

Full abstraction to understand graceful language interoperability.

Multi-language stories



Graceful interoperation?



Abstraction leaks?

(Several expert languages: not (yet?) in this work)

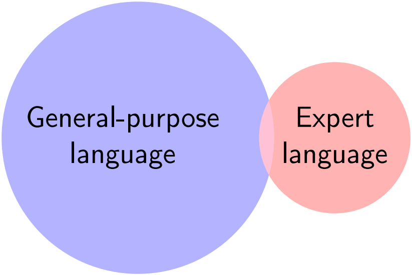
Full abstraction

$[\![\_]\!] : S \longrightarrow T$ fully abstract:

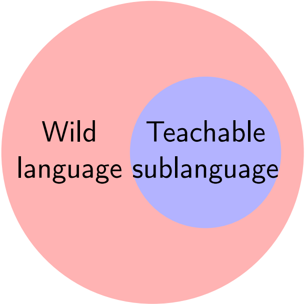
$$a \approx^{ctx} b \implies [\![a]\!] \approx^{ctx} [\![b]\!]$$

Full abstraction preserves (equational) reasoning.

Full abstraction for multi-language systems



Graceful interoperation: $G \xrightarrow{f.a.} (G + E)$



No abstraction leaks: $T \xrightarrow{f.a.} W$

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**.

Our case study

U (Unrestricted): general-purpose ML language
L (Linear): expert linear language.

$$U \xrightarrow{f.a.} (U + L)$$

Proof: by translating L back into U in an inefficient but correct way.

Our case study

U (Unrestricted): general-purpose ML language
L (Linear): expert linear language.

$$U \xrightarrow{f.a.} (U + L)$$

Proof: by translating L back into U in an inefficient but correct way.

Note: extending U preserves this result.

Our case study

U (Unrestricted): general-purpose ML language
L (Linear): expert linear language.

$$U \xrightarrow{f.a.} (U + L)$$

Proof: by translating L back into U in an inefficient but correct way.

Note: extending U preserves this result.

Note: $U \longrightarrow (U + L)$ not meant to be fully abstract.
(Not robust to extensions of U)

Section 2

Case Study: Unrestricted and Linear

Unrestricted language: syntax

Types	σ	$::=$	$\alpha \mid \sigma_1 \times \sigma_2 \mid 1 \mid \sigma_1 \to \sigma_2 \mid$ $\sigma_1 + \sigma_2 \mid \mu\alpha.\,\sigma \mid \forall\alpha.\,\sigma$
Expressions	e	$::=$	$x \mid$ $\langle e_1, e_2 \rangle \mid \pi_1\, e \mid \pi_2\, e \mid$ $\langle\rangle \mid e_1; e_2 \mid$ $\lambda(x : \sigma).\, e \mid e_1\ e_2 \mid$ $\mathsf{inj}_1\, e \mid \mathsf{inj}_2\, e \mid \mathsf{case}\, e'\, \mathsf{of}\, x_1.\, e_1 \mid x_2.\, e_2 \mid$ $\mathsf{fold}_{\mu\alpha.\sigma}\, e \mid \mathsf{unfold}\, e \mid$ $\Lambda\alpha.\, e \mid e\,[\sigma]$
Typing contexts	Γ, Δ	$::=$	$\cdot \mid \Gamma, x : \sigma \mid \Gamma, \alpha$

Linear types: introduction

Resource tracking, unique ownership.

σ $\quad$ $!\sigma$ $\quad$ Γ $\quad$ $!\Gamma$

$$\Gamma \vdash_l \mathbf{e} : \sigma$$

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

$$\begin{aligned}\sigma ::= \; & \sigma_1 \otimes \sigma_2 \mid \mathbf{1} \mid \sigma_1 \multimap \sigma_2 \mid \\ & \sigma_1 \oplus \sigma_2 \mid \mu\alpha.\, \sigma \mid \alpha \mid \\ & !\sigma \mid \\ & \mathbf{Box}\ \mathbf{b}\ \sigma\end{aligned}$$

Linear types: base

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

$$\frac{}{!\Gamma, \mathbf{x}:\sigma \vdash_l \mathbf{x} : \sigma} \qquad \frac{}{!\Gamma \vdash_l \langle\rangle : \mathbf{1}} \qquad \frac{\Gamma \vdash_l \mathbf{e} : \mathbf{1} \qquad \Delta \vdash_l \mathbf{e}' : \sigma}{\Gamma \curlyvee \Delta \vdash_l \mathbf{e}; \mathbf{e}' : \sigma}$$

$$\frac{\Gamma_1 \vdash_l \mathbf{e}_1 : \sigma_1 \qquad \Gamma_2 \vdash_l \mathbf{e}_2 : \sigma_2}{\Gamma_1 \curlyvee \Gamma_2 \vdash_l \langle \mathbf{e}_1, \mathbf{e}_2 \rangle : \sigma_1 \otimes \sigma_2} \qquad \frac{\Gamma \vdash_l \mathbf{e} : \sigma_1 \otimes \sigma_2 \qquad \Delta, \mathbf{x}_1 : \sigma_1, \mathbf{x}_2 : \sigma_2 \vdash_l \mathbf{e}' : \sigma}{\Gamma \curlyvee \Delta \vdash_l \mathbf{let}\ \langle \mathbf{x}_1, \mathbf{x}_2 \rangle = \mathbf{e}\ \mathbf{in}\ \mathbf{e}' : \sigma}$$

$$\frac{\Gamma, \mathbf{x}:\sigma \vdash_l \mathbf{e} : \sigma'}{\Gamma \vdash_l \lambda(\mathbf{x}:\sigma).\,\mathbf{e} : \sigma \multimap \sigma'} \qquad \frac{\Gamma \vdash_l \mathbf{e} : \sigma' \multimap \sigma \qquad \Delta \vdash_l \mathbf{e}' : \sigma'}{\Gamma \curlyvee \Delta \vdash_l \mathbf{e}\ \mathbf{e}' : \sigma}$$

$$\frac{\Gamma \vdash_l \mathbf{e} : \sigma_i}{\Gamma \vdash_l \mathbf{inj_i}\,\mathbf{e} : \sigma_1 \oplus \sigma_2} \qquad \frac{\Gamma \vdash_l \mathbf{e} : \sigma_1 \oplus \sigma_2 \qquad (\Delta, \mathbf{x}_i : \sigma_i \vdash_l \mathbf{e}_i : \sigma)_{\mathbf{i} \in \{\mathbf{1},\mathbf{2}\}}}{\Gamma \curlyvee \Delta \vdash_l \mathbf{case}\ \mathbf{e}\ \mathbf{of}\ \mathbf{x}_1.\,\mathbf{e}_1 \mid \mathbf{x}_2.\,\mathbf{e}_2 : \sigma}$$

$$\frac{!\Gamma \vdash_l \mathbf{e} : \sigma}{!\Gamma \vdash_l \mathbf{share}\ \mathbf{e} : !\sigma} \qquad \frac{\Gamma \vdash_l \mathbf{e} : !\sigma}{\Gamma \vdash_l \mathbf{copy}^{\sigma}\ \mathbf{e} : \sigma}$$

Applications

Protocol with resource handling requirements.

"This file descriptor must be closed"

$$\begin{array}{lcl} \textbf{open} & : & !(![\mathsf{Path}] \multimap \textbf{Handle}) \\ \textbf{line} & : & !(\textbf{Handle} \multimap (\textbf{Handle} \oplus (![\mathsf{String}] \otimes \textbf{Handle}))) \\ \textbf{close} & : & !(\textbf{Handle} \multimap \mathbf{1}) \end{array}$$

(details about the boundaries come later)

Typestate.

(details about the boundaries come later)

$$\begin{array}{lll} \textbf{open} & : & !(![\mathsf{Path}] \multimap \textbf{Handle}) \\ \textbf{line} & : & !(\textbf{Handle} \multimap (\textbf{Handle} \oplus (![\mathsf{String}] \otimes \textbf{Handle}))) \\ \textbf{close} & : & !(\textbf{Handle} \multimap \mathbf{1}) \end{array}$$

```
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)))
```

(U values are passed back and forth, never inspected)

Linear types: linear locations

Box $1\ \sigma$: full cell

Box $0\ \sigma$: empty cell

$$\mathbf{1} \;\underset{\textbf{free}}{\overset{\textbf{new}}{\substack{\multimap \\ \multimapinv}}}\; \textbf{Box}\ 0\ \sigma \qquad\qquad \textbf{Box}\ 1\ \sigma \;\underset{\textbf{box}}{\overset{\textbf{unbox}}{\substack{\multimap \\ \multimapinv}}}\; (\textbf{Box}\ 0\ \sigma) \otimes \sigma$$

Applications: in-place reuse of memory cells.

List reversal

```
type LList a = μt. 1 ⊕ Box 1 (a ⊗ t)
pattern Nil = inl ()
pattern Cons l x xs = inr (box (l, (x, xs)))

val reverse : LList a ⊸ LList a
let reverse list = loop Nil list
  where rec loop tail = function
  | Nil → tail
  | Cons l x xs → loop (Cons l x tail) xs

type List a = μt. 1 + (a × t)
let reverse list = UL(share (reverse (copy (LU(list)))))
```

(U values are created from the L side from a compatible type)

Interaction: lump

Types $\sigma \mid \boldsymbol{\sigma}$

$$\begin{aligned} &\sigma \\ &\boldsymbol{\sigma} \quad + ::= \cdots \mid [\sigma] \end{aligned}$$

Expressions $e \mid \mathbf{e}$

$$\begin{aligned} &e \quad + ::= \cdots \mid \mathcal{UL}(\mathbf{e}) \\ &\mathbf{e} \quad + ::= \cdots \mid \mathcal{LU}(e) \end{aligned}$$

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

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

Interaction: compatibility

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

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

$$\frac{\vdash_{\mathsf{ul}} \sigma_1 \simeq !\sigma_1 \qquad \vdash_{\mathsf{ul}} \sigma_2 \simeq !\sigma_2}{\vdash_{\mathsf{ul}} \sigma_1 + \sigma_2 \simeq !(\sigma_1 \oplus \sigma_2)} \qquad \frac{\vdash_{\mathsf{ul}} \sigma \simeq !\sigma \qquad \vdash_{\mathsf{ul}} \sigma' \simeq !\sigma'}{\vdash_{\mathsf{ul}} \sigma \to \sigma' \simeq !(!\sigma \multimap !\sigma')}$$

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

Interaction primitives and derived constructs:

$$![\sigma] \; \underset{\mathsf{lump}^{\sigma}}{\overset{{}^{\sigma}\mathsf{unlump}}{\rightleftarrows}} \; \sigma \quad \text{when} \quad \vdash_{\mathsf{ul}} \sigma \simeq \sigma$$

$${}^{\sigma}\mathcal{LU}(e) \stackrel{\text{def}}{=} {}^{\sigma}\mathsf{unlump}\ \mathcal{LU}(e)$$

$$\mathcal{UL}^{\sigma}(\mathbf{e}) \stackrel{\text{def}}{=} \mathcal{UL}(\mathsf{lump}^{\sigma}\ \mathbf{e})$$

Full abstraction

Theorem

The embedding of U *into* UL *is fully abstract.*

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

Full abstraction

Theorem

The embedding of U into UL is fully abstract.

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

$$
\begin{aligned}
\lceil !\sigma \rceil &\stackrel{\text{def}}{=} \lceil \sigma \rceil \\
\lceil \textbf{Box}\ 0\ \sigma \rceil &\stackrel{\text{def}}{=} 1 \\
\lceil \textbf{Box}\ 1\ \sigma \rceil &\stackrel{\text{def}}{=} 1 \times \lceil \sigma \rceil \\
\lceil \sigma_1 \otimes \sigma_2 \rceil &\stackrel{\text{def}}{=} \lceil \sigma_1 \rceil \times \lceil \sigma_2 \rceil
\end{aligned}
$$

(Cogent)

Remark on parametricity

(from Max New)

$$(\Lambda\alpha.\,\lambda(x:\alpha).\,\mathcal{U}\mathcal{L}^{\alpha}({}^{\alpha}\mathcal{L}\mathcal{U}(x)))\;[\sigma] \quad \overset{U}{\hookrightarrow}? \quad \lambda(x:\sigma).\,\mathcal{U}\mathcal{L}^{\sigma}({}^{\sigma}\mathcal{L}\mathcal{U}(x))$$

Not well-typed!

$$(\Lambda\alpha.\,\lambda(x:\alpha).\,\mathcal{U}\mathcal{L}^{![\alpha]}({}^{![\alpha]}\mathcal{L}\mathcal{U}(x)))\;[\sigma] \quad \overset{U}{\hookrightarrow} \quad \lambda(x:\sigma).\,\mathcal{U}\mathcal{L}^{![\sigma]}({}^{![\sigma]}\mathcal{L}\mathcal{U}(x))$$

Questions

But not the end!

Implementation ?

Implementability? (Cell size.)

Limitation: no separation of pointer and capability.

Does this approach scale to a language usable in practice?
(Polymorphism in **L**?)
(Without losing its simplicity?)

Your questions.

Section 3

How Fully Abstract Can We Go?

I used to think of Full Abstraction as an **ideal** property that would never be reached in practice.

I changed my mind. The statement can be **weakened** to fit many situations, and remains a useful specification.

I will now present some (abstract) examples of this approach.

Weak Trick 1: restrict the interaction types

The no-interaction multi-language: always fully abstract!

Types restrict interaction: "only integers", "only ground types".

Extend the scope of safe interaction by adding more types.
Design tool.

Idea: the idealist will still have a useful system.

Weak Trick 2: weaken the source equivalence

Full abstraction is relative to the source equivalence.

Contextual equivalence makes a closed-world assumption.
Good, sometimes too strong.

Safe impure language: forbid reordering of calls.

Safe impure language: add impure counters for user reasoning.

Or use types with weaker equivalence principles: **linking types**
(Daniel Patterson, Amal Ahmed)

Idea: full abstraction forces you to **specify** the right thing.

Questions

Compare different ways to specify a weaker equivalence for full abstraction?

- through explicit term equations?
- through types?
- by adding phantom features?

Does our multi-language design scale to more than two languages?
(Yes, I think)

Are boundaries multi-language designs also convenience boundaries?
(good or bad?)

Your questions.

Thanks!