Full abstraction for multi-language systems
ML plus linear types

Gabriel Scherer, Amal Ahmed, Max New

Northeastern University, Boston

February 9, 2017

2. Case Study: Unrestricted and Linear

3. How Fully Abstract Can We Go?
Section 1

Full Abstraction for Multi-Language Systems: Introduction
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

- General-purpose language
- Expert language

Graceful interoperation?

- Wild language
- Teachable sublanguage

Abstraction leaks?

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

\([\_] : S \rightarrow T\) fully abstract:

\[a \approx_{\text{ctx}} b \implies \llbracket a \rrbracket \approx_{\text{ctx}} \llbracket b \rrbracket\]

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.

Note: extending \( U \) preserves this result.

Note: \( U \xrightarrow{\text{not meant to be fully abstract.}} (U + L) \) not meant to be fully abstract.

(Not robust to extensions of \( U \))
Our case study

**U** (Unrestricted): general-purpose ML language

**L** (Linear): expert linear language.

\[ U \overset{f.a.}{\rightarrow} (U + L) \]

**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: \( U \rightarrow (U + L) \) not meant to be fully abstract.
(Not robust to extensions of \( U \))
Section 2

Case Study: Unrestricted and Linear
Unrestricted language: syntax

**Types**

\[ \sigma ::= \alpha \mid \sigma_1 \times \sigma_2 \mid 1 \mid \sigma_1 \to \sigma_2 \mid \sigma_1 + \sigma_2 \mid \mu \alpha. \sigma \mid \forall \alpha. \sigma \]

**Expressions**

\[ e ::= x \mid \langle e_1, e_2 \rangle \mid \pi_1 e \mid \pi_2 e \mid \langle \rangle \mid e_1 ; e_2 \mid \lambda (x : \sigma). e \mid e_1 e_2 \mid \text{inj}_1 e \mid \text{inj}_2 e \mid \text{case } e' \text{ of } x_1. e_1 \mid x_2. e_2 \mid \text{fold}_{\mu \alpha. \sigma} e \mid \text{unfold } e \mid \Lambda \alpha. e \mid e [\sigma] \]

**Typing contexts**

\[ \Gamma, \Delta ::= \cdot \mid \Gamma, x : \sigma \mid \Gamma, \alpha \]
Linear types: introduction

Resource tracking, unique ownership.

\[
\sigma \quad \! \sigma \quad \Gamma \quad \! \Gamma \\
\Gamma \vdash_1 e : \sigma
\]

We own \(e\) at type \(\sigma\) (duplicable or not), \(e\) owns the resources in \(\Gamma\).

\[
\sigma ::= \sigma_1 \otimes \sigma_2 \mid 1 \mid \sigma_1 \multimap \sigma_2 \mid \\
\sigma_1 \oplus \sigma_2 \mid \mu \alpha. \sigma \mid \alpha \mid \\
\! \sigma \mid \\
\text{Box } b \ \sigma
\]
Linear types: base

A simple but useful language with linear types.

\[ \Gamma, x : \sigma \vdash x : \sigma \]
\[ !\Gamma \vdash \langle \rangle : 1 \]
\[ !\Gamma \vdash e : 1 \quad \Delta \vdash e' : \sigma \]
\[ \Gamma \vdash \langle e_1, e_2 \rangle : \sigma_1 \otimes \sigma_2 \]
\[ \Delta, x_1 : \sigma_1, x_2 : \sigma_2 \vdash e' : \sigma \]
\[ \Gamma \vdash \langle x_1, x_2 \rangle = e \text{ in } e' : \sigma \]
\[ \Gamma \vdash \lambda(x : \sigma). e : \sigma \to \sigma' \]
\[ \Gamma \vdash \lambda e' : \sigma' \rightarrow \sigma \]
\[ \Delta \vdash e' : \sigma' \]
\[ \Gamma \vdash e : \sigma \quad \Delta \vdash e \text{ of } x_1. e_1 | x_2. e_2 : \sigma \]
\[ \Gamma \vdash inj_i e : \sigma_1 \oplus \sigma_2 \]
\[ \Gamma \vdash e : \sigma_1 \oplus \sigma_2 \]
\[ (\Delta, x_j : \sigma_j \vdash e_j : \sigma)_{i \in \{1, 2\}} \]
\[ \Gamma \vdash \text{case } e \text{ of } x_1. e_1 | x_2. e_2 : \sigma \]
\[ !\Gamma \vdash e : \sigma \]
\[ !\Gamma \vdash \text{share } e : !\sigma \]
\[ !\Gamma \vdash \text{copy}^\sigma e : \sigma \]
Applications

Protocol with resource handling requirements.

“This file descriptor must be closed”

\[
\begin{align*}
\text{open} & : ![\text{Path} \rightarrow \text{Handle}] \\
\text{line} & : ![\text{Handle} \rightarrow (\text{Handle} \oplus (![\text{String}] \otimes \text{Handle}))] \\
\text{close} & : ![\text{Handle} \rightarrow 1]
\end{align*}
\]

(details about the boundaries come later)

Typestate.
open : ![Path] → Handle
line : ![Handle] → (Handle ⊕ ([String] ⊗ Handle))
close : ![Handle] → 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)))

(U values are passed back and forth, never inspected)
Linear types: linear locations

**Box 1 \( \sigma \):** full cell

**Box 0 \( \sigma \):** empty cell

---

![Diagram](https://via.placeholder.com/150)

- new
- free

**Applications:** in-place reuse of memory cells.
List reversal

type LList a = \( \mu t. 1 \oplus \text{Box } 1 \ (a \otimes t) \)

pattern Nil = \text{inl } ()

pattern Cons l x xs = \text{inr } (\text{box } (l, (x, xs)))

val reverse : LList a \(\rightarrow\) LList a

let reverse list = loop Nil list

where rec loop tail = function
  | Nil \rightarrow tail
  | Cons l x xs \rightarrow loop (Cons l x tail) xs


val reverse : LList a \(\rightarrow\) LList a

let reverse list = UL(share (reverse (copy (LU(list)))))

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

**Types** \( \sigma \mid \sigma \)

\[
\begin{align*}
\sigma & \\
\sigma + & ::= \cdots \mid [\sigma]
\end{align*}
\]

**Expressions** \( e \mid e \)

\[
\begin{align*}
e & + ::= \cdots \mid \mathcal{UL}(e) \\
e & + ::= \cdots \mid \mathcal{LU}(e)
\end{align*}
\]

**Contexts** \( \Gamma ::= \cdot \mid \Gamma, x : \sigma \mid \Gamma, \alpha \mid \Gamma, x : \sigma \)

\[
\begin{align*}
!\Gamma \vdash_{lu} e : \sigma & \\
!\Gamma \vdash_{ul} \mathcal{LU}(e) : ![\sigma]
\end{align*}
\]

\[
\begin{align*}
!\Gamma \vdash_{ul} e : ![\sigma] & \\
!\Gamma \vdash_{lu} \mathcal{UL}(e) : \sigma
\end{align*}
\]
Interaction: compatibility

Compatibility relation \( \vdash_{ul} \sigma \simeq \sigma \)

\[
\vdash_{ul} 1 \simeq !1
\]

\[
\vdash_{ul} \sigma_1 \simeq !\sigma_1 \quad \vdash_{ul} \sigma_2 \simeq !\sigma_2
\]

\[
\vdash_{ul} \sigma_1 \times \sigma_2 \simeq !(\sigma_1 \otimes \sigma_2)
\]

\[
\vdash_{ul} \sigma_1 \simeq !\sigma_1 \quad \vdash_{ul} \sigma_2 \simeq !\sigma_2
\]

\[
\vdash_{ul} \sigma_1 + \sigma_2 \simeq !(\sigma_1 \oplus \sigma_2)
\]

\[
\vdash_{ul} \sigma \simeq ![\sigma]
\]

\[
\vdash_{ul} \sigma \simeq !!\sigma
\]

\[
\vdash_{ul} \sigma \simeq !(\text{Box 1 } \sigma)
\]

Interaction primitives and derived constructs:

\( ^{\sigma}\text{unlump} \)

\[
![\sigma] \quad \quad \sigma \quad \quad \text{when} \quad \quad \vdash_{ul} \sigma \simeq \sigma
\]

\( \sigma \text{LU}(e) \overset{\text{def}}{=} ^{\sigma}\text{unlump LU}(e) \)

\( \text{UL}^\sigma(e) \overset{\text{def}}{=} \text{UL}(\text{lump}^\sigma e) \)
Theorem

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

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

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

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

$$
\begin{align*}
[!\sigma] & \quad \overset{\text{def}}{=} \quad [\sigma] \\
[\text{Box} \ 0 \ \sigma] & \quad \overset{\text{def}}{=} \quad 1 \\
[\text{Box} \ 1 \ \sigma] & \quad \overset{\text{def}}{=} \quad 1 \times [\sigma] \\
[\sigma_1 \otimes \sigma_2] & \quad \overset{\text{def}}{=} \quad [\sigma_1] \times [\sigma_2]
\end{align*}
$$

(Cogent)
Remark on parametricity

(from Max New)

\[(\Lambda \alpha. \lambda (x : \alpha). U \mathcal{L}^{\alpha}(\alpha \mathcal{L} U (x))) [\sigma] \overset{U}{\rightarrow} ? \lambda (x : \sigma). U \mathcal{L}^{\sigma}(\sigma \mathcal{L} U (x))\]

Not well-typed!

\[(\Lambda \alpha. \lambda (x : \alpha). U \mathcal{L}^{[\alpha]}([\alpha] \mathcal{L} U (x))) [\sigma] \overset{U}{\rightarrow} \lambda (x : \sigma). U \mathcal{L}^{[\sigma]}([\sigma] \mathcal{L} U (x))\]
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!