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

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

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

Full abstraction preserves (equational) reasoning.

Full abstraction for multi-language systems

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

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

Full abstraction preserves (equational) reasoning.

$$\begin{tikzcd}
S_1 \arrow[dr, "\text{full abs.}"'] & & S_2 \arrow[dl, "\text{full abs.}"] \\
& T &
\end{tikzcd}$$

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

Full abstraction for multi-language systems

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

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

Full abstraction preserves (equational) reasoning.

$$\begin{tikzcd}
S_1 \arrow[rr, squiggly, dash, "\text{interop.}"] \arrow[dr, "\text{full abs.}"'] & & S_2 \arrow[dl, "\text{full abs.}"] \\
& T &
\end{tikzcd}$$

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.

$$\frac{\Gamma_1 \vdash \sigma_1 \qquad \Gamma_2 \vdash \sigma_2}{\Gamma_1 \curlyvee \Gamma_2 \vdash \sigma_1 \otimes \sigma_2}$$

$$\frac{\Gamma \vdash \sigma_1 \otimes \sigma_2 \qquad \Delta, \sigma_1, \sigma_2 \vdash \sigma}{\Gamma \curlyvee \Delta \vdash \sigma}$$

$$\frac{}{!\Gamma \vdash \mathbf{1}} \qquad \frac{\Gamma \vdash \mathbf{1} \qquad \Delta \vdash \sigma}{\Gamma \curlyvee \Delta \vdash \sigma}$$

$$\frac{\Gamma, \sigma \vdash \sigma'}{\Gamma \vdash \sigma \multimap \sigma'} \qquad \frac{\Gamma \vdash \sigma' \multimap \sigma \qquad \Delta \vdash \sigma'}{\Gamma \curlyvee \Delta \vdash \sigma}$$

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

Linear types: base

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

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

$$\frac{\Gamma \vdash e : \sigma_1 \otimes \sigma_2 \qquad \Delta, x_1 : \sigma_1, x_2 : \sigma_2 \vdash e' : \sigma}{\Gamma \curlyvee \Delta \vdash \mathbf{let}\ \langle x_1, x_2 \rangle = e\ \mathbf{in}\ e' : \sigma}$$

$$\frac{}{!\Gamma \vdash \langle\rangle : 1} \qquad \frac{\Gamma \vdash e : 1 \qquad \Delta \vdash e' : \sigma}{\Gamma \curlyvee \Delta \vdash e; e' : \sigma}$$

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

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

Linear types: base

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

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

$$\frac{\Gamma \vdash e : \sigma_1 \otimes \sigma_2 \qquad \Delta, x_1 : \sigma_1, x_2 : \sigma_2 \vdash e' : \sigma}{\Gamma \curlyvee \Delta \vdash \mathbf{let}\ \langle x_1, x_2 \rangle = e\ \mathbf{in}\ e' : \sigma}$$

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

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

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

Linear types: base

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

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

$$\frac{\Gamma \vdash e : \sigma_1 \otimes \sigma_2 \qquad \Delta, x_1 : \sigma_1, x_2 : \sigma_2 \vdash e' : \sigma}{\Gamma \curlyvee \Delta \vdash \mathbf{let}\ \langle x_1, x_2 \rangle = e\ \mathbf{in}\ e' : \sigma}$$

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

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

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

Applications

Protocol with resource handling requirements.

"This file descriptor must be closed"

Typestate.

Linear types: linear locations

$\textbf{Box 1 } \sigma$: full cell

$\textbf{Box 0 } \sigma$: empty cell

$$\mathbf{1} \;\underset{\textbf{free}}{\overset{\textbf{new}}{\rightleftarrows}}\; \textbf{Box 0 } \sigma \qquad\qquad \textbf{Box 1 } \sigma \;\underset{\textbf{box}}{\overset{\textbf{unbox}}{\rightleftarrows}}\; \textbf{Box 0 } \sigma \otimes \sigma$$

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 (Conx l x tail) xs

(* use reverse internally *)

(* on the ML side *)
type List a = μt. 1 + (a × 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.

$$\mathcal{LU}\,(\sigma) \quad \stackrel{\text{def}}{=} \quad \mathbf{!}\lfloor \sigma \rfloor \qquad \begin{aligned} \lfloor \sigma_1 \times \sigma_2 \rfloor &\stackrel{\text{def}}{=} \mathbf{Box\ 1}\,(\lfloor \sigma_1 \rfloor \otimes \lfloor \sigma_2 \rfloor) \\ \lfloor 1 \rfloor &\stackrel{\text{def}}{=} \mathbf{1} \\ \lfloor \sigma_1 \to \sigma_2 \rfloor &\stackrel{\text{def}}{=} \mathcal{LU}\,(\sigma_1) \multimap \mathcal{LU}\,(\sigma_2) \end{aligned}$$

This gives a direct multi-language semantics.

Full abstraction

The ML language can be compiled into a linear language.

$$\mathcal{LU}(\sigma) \stackrel{\text{def}}{=} \mathbf{!}\lfloor \sigma \rfloor \qquad \begin{array}{lcl} \lfloor \sigma_1 \times \sigma_2 \rfloor & \stackrel{\text{def}}{=} & \mathbf{Box\ 1}\ (\lfloor \sigma_1 \rfloor \otimes \lfloor \sigma_2 \rfloor) \\ \lfloor 1 \rfloor & \stackrel{\text{def}}{=} & \mathbf{1} \\ \lfloor \sigma_1 \to \sigma_2 \rfloor & \stackrel{\text{def}}{=} & \mathcal{LU}(\sigma_1) \multimap \mathcal{LU}(\sigma_2) \end{array}$$

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.

$$\mathcal{LU}\,(\sigma) \overset{\text{def}}{=} \mathbf{!}\lfloor\sigma\rfloor$$

$$\begin{aligned}
\lfloor \sigma_1 \times \sigma_2 \rfloor &\overset{\text{def}}{=} \mathbf{Box\ 1}\,(\lfloor\sigma_1\rfloor \otimes \lfloor\sigma_2\rfloor) \\
\lfloor 1 \rfloor &\overset{\text{def}}{=} \mathbf{1} \\
\lfloor \sigma_1 \to \sigma_2 \rfloor &\overset{\text{def}}{=} \mathcal{LU}\,(\sigma_1) \multimap \mathcal{LU}\,(\sigma_2)
\end{aligned}$$

This gives a direct multi-language semantics.
Full abstraction by pure interpretation of the linear language in ML.

$$\begin{aligned}
\lceil \mathbf{!\sigma} \rceil &\overset{\text{def}}{=} \lceil \boldsymbol{\sigma} \rceil \\
\lceil \mathbf{Box\ 1\ \sigma} \rceil &\overset{\text{def}}{=} \lceil \boldsymbol{\sigma} \rceil \\
\lceil \mathbf{Box\ 0\ \sigma} \rceil &\overset{\text{def}}{=} 1 \\
\lceil \boldsymbol{\sigma}_1 \otimes \boldsymbol{\sigma}_2 \rceil &\overset{\text{def}}{=} \lceil \boldsymbol{\sigma}_1 \rceil \times \lceil \boldsymbol{\sigma}_2 \rceil
\end{aligned}$$

(Cogent)

Going further

Polymorphism not formalized yet.

Implementation?