Semantic Intermediate Representations for Sound Language Interoperability

Amal Ahmed

Northeastern University

Joint work with: Daniel Patterson, Andrew Wagner, Noble Mushtak

Type Soundness: In Theory

ML Haskell Java

Metatheoreticians:

verify type soundness of at least a core of the language,

usually via progress and preservation

Type Soundness: In Practice?

escape hatches


ctypes


inline-c

JNI

Working Metatheoreticians:

(those doing metatheory of langs as they are implemented)

what techniques can they use for soundness of lang+interop?

Interoperability in Practice

All languages support interoperability:

- C FFI (with tools like SWIG to generate wrappers)

- OCaml-ctypes, inline-c (C in Haskell), Java JNI, Rust bindgen

- Interop between languages that target JVM:
  - Java, Scala, Clojure, Kotlin



- Interop between .NET languages that target CIL:
  - C#, F#, VB, etc.


Type Soundness + Interoperability

- *Matthews-Findler POPL'07*: Multi-language Semantics

$$\begin{array}{llll} Lang\ A & \mathbf{e} & ::= & \ldots \mid {}^{\tau}AB^{\tau}\,e \\ Lang\ B & e & ::= & \ldots \mid {}^{\tau}BA^{\tau}\,\mathbf{e} \end{array}$$

- Give typing rules for boundary terms
- Give type-directed operational semantics to boundary terms

$${}^{\tau}AB^{\tau}\,e \longrightarrow \ldots \longrightarrow {}^{\tau}AB^{\tau}\,v \longrightarrow \mathbf{v}$$

$${}^{\tau}BA^{\tau}\,\mathbf{e} \longrightarrow \ldots \longrightarrow {}^{\tau}BA^{\tau}\,\mathbf{v} \longrightarrow v$$

Type Soundness + Interoperability

- *Matthews-Findler POPL'07*: Multi-language Semantics

$$\begin{array}{llll} Lang\ A & \mathbf{e} & ::= & \ldots \mid {}^{\tau}AB^{\tau}\ e \\ Lang\ B & e & ::= & \ldots \mid {}^{\tau}BA^{\tau}\ \mathbf{e} \end{array}$$

- Give typing rules for boundary terms
- Give type-directed operational semantics to boundary terms

$${}^{\tau}AB^{\tau}\ e \longrightarrow \ldots \longrightarrow {}^{\tau}AB^{\tau}\ v \longrightarrow \mathbf{v}$$

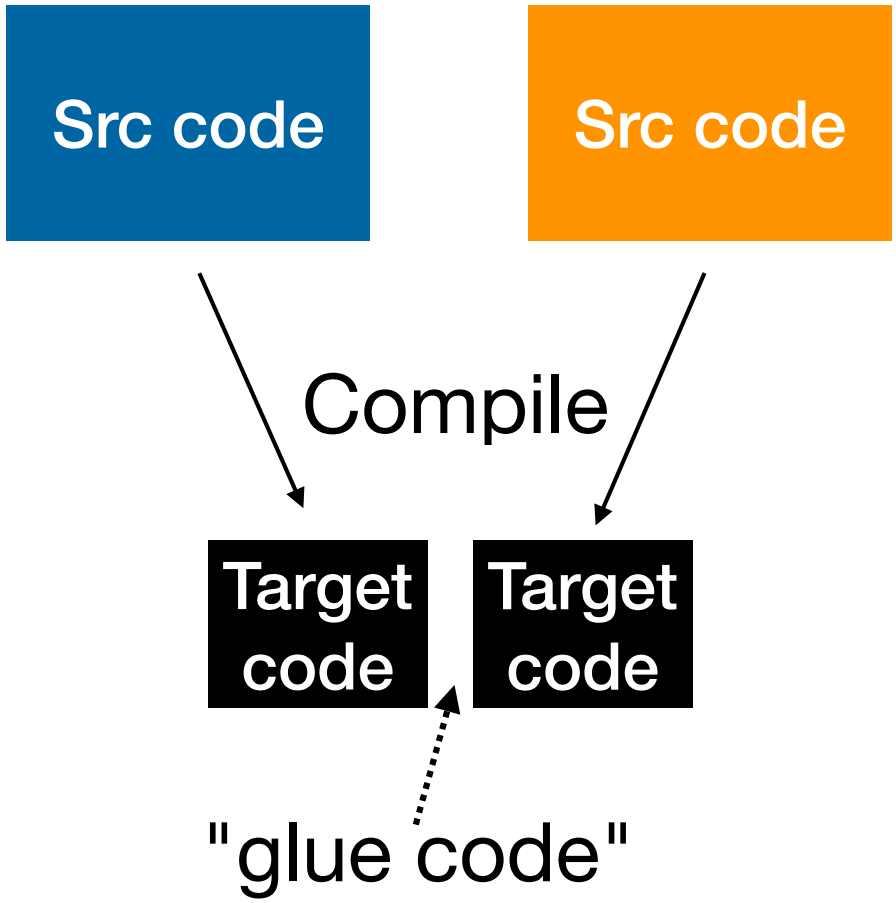
$${}^{\tau}BA^{\tau}\ \mathbf{e} \longrightarrow \ldots \longrightarrow {}^{\tau}BA^{\tau}\ \mathbf{v} \longrightarrow v$$

- Prove type soundness of multi-language

Type Soundness + Interoperability

- *Matthews-Findler POPL'07*: Multi-language Semantics
- Has been widely applied, e.g., for interop between:
  - simple & dependent types
    [Osera-Sjoberg-Zdancewic PLPV'12]
  - unrestricted & substructural types
    [Tov-Pucella ESOP10, Scherer et al. FOSSACS'18]
  - high-level functional language & assembly
    [Patterson et al. PLDI'17]
  - compiler source & target languages
    [Ahmed-Blume ICFP'11, Perconti-Ahmed ESOP'14, New et al. ICFP'16]
- *But it's not how FFIs are implemented in practice!*

Interoperability in Practice



Needed: Framework for design & verification of sound language interoperability that is connected to implementation

Interoperability Soundness via Semantic Intermediate Representations

Recipe for Sound Language Interop.

Recipe for Sound Language Interop.

$e : \tau_A$ $\quad e : \tau_B$

Add boundary forms: $\langle\!| e |\!\rangle_{\tau_A}$ $\quad \langle\!| e |\!\rangle_{\tau_B}$

convertibility

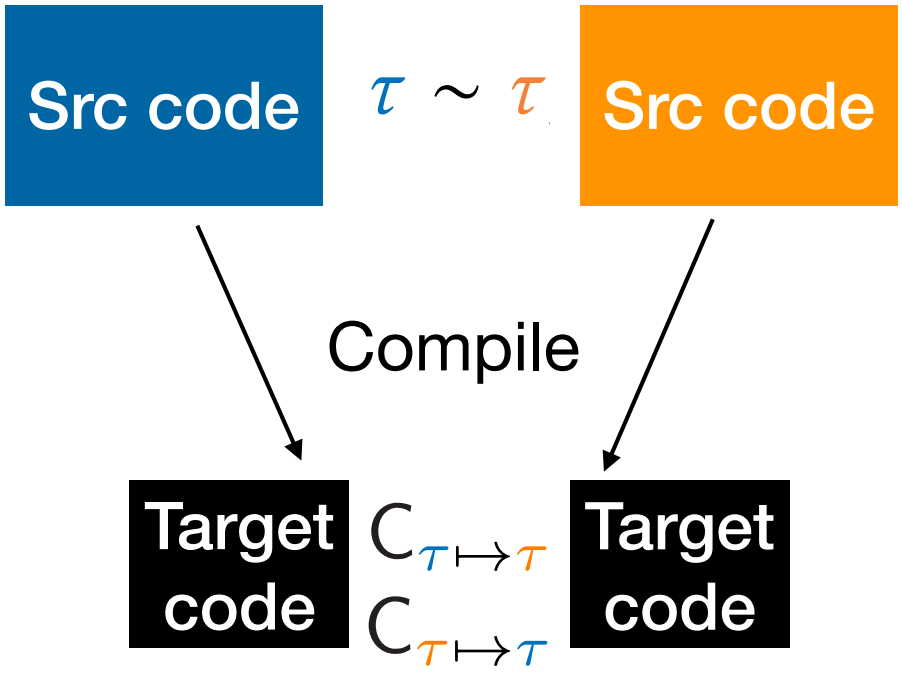
$$\frac{\Gamma;\Gamma \vdash e : \tau_B \quad \tau_B \sim \tau_A}{\Gamma;\Gamma \vdash \langle\!| e |\!\rangle_{\tau_A} : \tau_A} \qquad \frac{\Gamma;\Gamma \vdash e : \tau_A \quad \tau_B \sim \tau_A}{\Gamma;\Gamma \vdash \langle\!| e |\!\rangle_{\tau_B} : \tau_B}$$

Recipe for Sound Language Interop.

1. Specify convertibility rules : $\tau_A \sim \tau_B$

Recipe for Sound Language Interop.

1. Specify convertibility rules : $\tau_A \sim \tau_B$
2. Implement target-level conversions for each pair of convertible types : $C_{\tau \mapsto \tau}(\cdot)$ / $C_{\tau \mapsto \tau}(\cdot)$



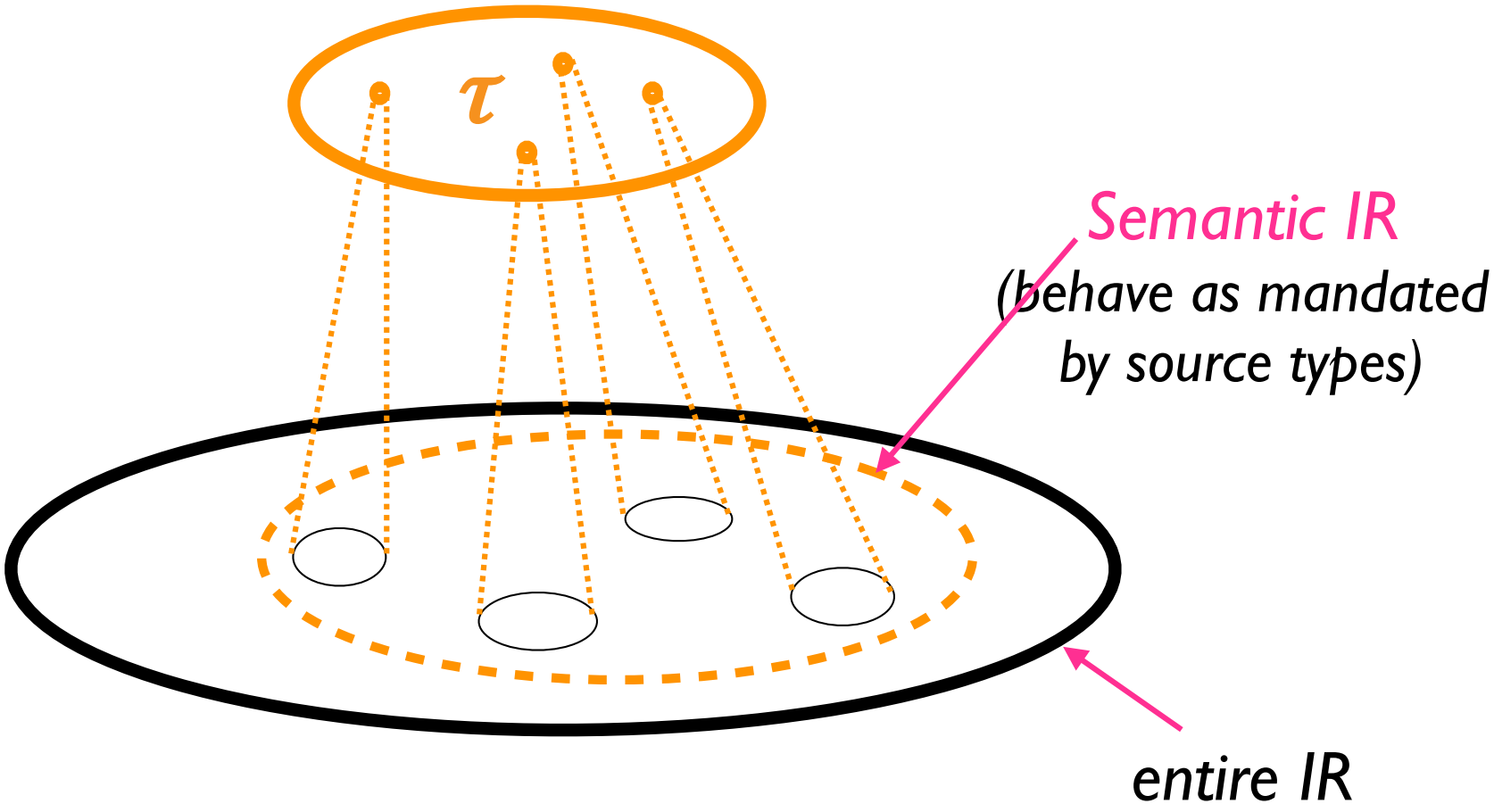
Recipe for Sound Language Interop.

1. Specify convertibility rules : $\tau_A \sim \tau_B$
2. Implement target-level conversions for each pair of convertible types : $C_{\tau \mapsto \tau}(\cdot)$ / $C_{\tau \mapsto \tau}(\cdot)$

To prove soundness:

3. Define semantic IR: $[\![\tau]\!] = \{e \mid \ldots\}$ $[\![\tau]\!] = \{e \mid \ldots\}$

logical relation: indexed by source types, inhabited by target terms


τ
Semantic IR
(behave as mandated
by source types)
entire IR

Recipe for Sound Language Interop.

1. Specify convertibility rules : $\tau_A \sim \tau_B$
2. Implement target-level conversions for each pair of convertible types : $C_{\tau \mapsto \tau}(\cdot)$ / $C_{\tau \mapsto \tau}(\cdot)$

inhabited by target terms

To prove soundness:

3. Define semantic IR: $[\![\tau]\!] = \{e \mid \ldots\}$ $[\![\tau]\!] = \{e \mid \ldots\}$
4. Prove conversion soundness: if $\tau \sim \tau$ then
   i. $e \in [\![\tau]\!] \implies C_{\tau \mapsto \tau}(e) \in [\![\tau]\!]$
   ii. $e \in [\![\tau]\!] \implies C_{\tau \mapsto \tau}(e) \in [\![\tau]\!]$

Recipe for Sound Language Interop.

1. Specify convertibility rules : $\tau_A \sim \tau_B$
2. Implement target-level conversions for each pair of convertible types : $C_{\tau \mapsto \tau}(\cdot)$ / $C_{\tau \mapsto \tau}(\cdot)$

inhabited by target terms

To prove soundness:

3. Define semantic IR: $[\![\tau]\!] = \{e \mid \ldots\}$ $[\![\tau]\!] = \{e \mid \ldots\}$
4. Prove conversion soundness: if $\tau \sim \tau$ then
   i. $e \in [\![\tau]\!] \implies C_{\tau \mapsto \tau}(e) \in [\![\tau]\!]$
   ii. $e \in [\![\tau]\!] \implies C_{\tau \mapsto \tau}(e) \in [\![\tau]\!]$
5. Prove type soundness:

$[\![\Gamma; \Gamma \vdash e : \tau]\!]$ and $[\![\Gamma; \Gamma \vdash e : \tau]\!]$

PLDI 2022

Semantic Soundness for Language Interoperability

Daniel Patterson
Northeastern University
Boston, MA, USA
dbp@dbpmail.net

Noble Mushtak
Northeastern University
Boston, MA, USA
mushtak.n@northeastern.edu

Andrew Wagner
Northeastern University
Boston, MA, USA
ahwagner@ccs.neu.edu

Amal Ahmed
Northeastern University
Boston, MA, USA
amal@ccs.neu.edu

Abstract

Programs are rarely implemented in a single language, and thus questions of type soundness should address not only the semantics of a single language, but how it interacts with others. Even between type-safe languages, disparate features can frustrate interoperability, as invariants from one language can easily be violated in the other. In their seminal 2007 paper, Matthews and Findler [33] proposed a multi-language construction that augments the interoperating languages with a pair of *boundaries* that allow code from one language to be embedded in the other. While this technique has been widely applied, their syntactic source-level interoperability doesn't reflect practical implementations, where the behavior of interaction is only defined after compilation to a common target, and any safety must be ensured by target invariants or inserted target-level "glue code."

In this paper, we present a novel framework for the design and verification of sound language interoperability that follows an interoperation-after-compilation strategy. Language

via a series of case studies that demonstrate how our semantic interoperation-after-compilation approach allows us both to account for complex differences in language semantics and make efficiency trade-offs based on particularities of compilers or targets.

CCS Concepts: • **Software and its engineering → General programming languages**.

Keywords: language interoperability, type soundness, semantics, logical relations

ACM Reference Format:
Daniel Patterson, Noble Mushtak, Andrew Wagner, and Amal Ahmed. 2022. Semantic Soundness for Language Interoperability. In *Proceedings of the 43rd ACM SIGPLAN International Conference on Programming Language Design and Implementation (PLDI '22), June 13–17, 2022, San Diego, CA, USA.* ACM, New York, NY, USA, 16 pages. https://doi.org/10.1145/3519939.3523703

1 Introduction

Case Studies

1. Shared Memory Interoperability & Data Representation
   - when is it safe to share mutable references across languages without copying or wrapping

Case Studies

1. Shared Memory Interoperability & Data Representation
   - when is it safe to share mutable references across languages without copying or wrapping
2. Unrestricted & Affine Languages: MiniML & Affi
   - safe mixing using runtime checks to ensure that affine resources are used at most once (but dynamic checks only when we don't have static assurance of affine use)

Case Studies

1. Shared Memory Interoperability & Data Representation
   - when is it safe to share mutable references across languages without copying or wrapping
2. Unrestricted & Affine Languages: MiniML & Affi
   - safe mixing using runtime checks to ensure that affine resources are used at most once (but dynamic checks only when we don't have static assurance of affine use)
3. Different Memory Management: MiniML & L^3
   - safe interop between languages with garbage collected and manually managed memory with safe strong updates

Case Study
Unrestricted & Affine: MiniML & Affi

Case Study: MiniML & Affi

MiniML : a polymorphic language with references

$$
\begin{array}{lll}
\tau & := & \texttt{unit} \mid \texttt{int} \mid \tau \times \tau \mid \tau + \tau \mid \tau \to \tau \mid \forall\alpha.\tau \mid \alpha \mid \texttt{ref}\ \tau \\
\texttt{e} & := & () \mid \texttt{n} \mid \texttt{x} \mid (\texttt{e},\texttt{e}) \mid \texttt{fst e} \mid \texttt{snd e} \mid \texttt{inl e} \mid \texttt{inr e} \mid \texttt{match e x}\{\texttt{e}\}\ \texttt{y}\{\texttt{e}\} \mid \\
 & & \lambda\texttt{x} : \tau.\texttt{e} \mid \texttt{e e} \mid \Lambda\alpha.\texttt{e} \mid \texttt{e}\!<\!\tau\!> \mid \texttt{ref e} \mid \texttt{!e} \mid \texttt{e} := \texttt{e} \mid \langle\!|\texttt{e}|\!\rangle_{\tau}
\end{array}
$$

Typing: $\Delta; \Gamma \vdash \texttt{e} : \tau$

boundaries

Affi : an affine language

$$
\begin{array}{lll}
\tau & ::= & \mathbf{unit} \mid \mathbf{int} \mid \tau \multimap \tau \mid !\tau \mid \tau \& \tau \mid \tau \otimes \tau \\
\mathbf{e} & ::= & () \mid \mathbf{n} \mid \mathbf{x} \mid \mathbf{a} \mid \lambda \mathbf{a} : \tau.\mathbf{e} \mid \mathbf{e}\ \mathbf{e} \mid !\mathbf{e} \mid \mathbf{let}\ !\mathbf{x} = \mathbf{e}\ \mathbf{in}\ \mathbf{e}' \mid \\
 & & \langle \mathbf{e}, \mathbf{e}' \rangle \mid \mathbf{e}.1 \mid \mathbf{e}.2 \mid (\mathbf{e}, \mathbf{e}) \mid \mathbf{let}\ (\mathbf{a}, \mathbf{a}') = \mathbf{e}\ \mathbf{in}\ \mathbf{e}' \mid \langle\!|\mathbf{e}|\!\rangle_{\tau}
\end{array}
$$

Typing: $\mathbf{\Gamma}; \mathbf{\Omega} \vdash \mathbf{e} : \boldsymbol{\tau}$

Case Study: MiniML & Affi

Target : untyped λ calculus with references & fail

LCVM

Expr e ::= () | n | ℓ | x | (e, e) | fst e | snd e | inl e | inr e
| if e {e} {e} | match e x{e} y{e} | let x = e in e
| λx{e} | e e | ref e | !e | e := e | fail c

Values v ::= () | n | ℓ | (v, v) | λx.e

Err c ::= Type | Conv

Affi Typing Rules

$$\boxed{\Gamma; \Omega \vdash e : \tau}$$

$$\frac{a : \tau \in \Omega}{\Gamma; \Omega \vdash a : \tau} \qquad \frac{x : \tau \in \Gamma}{\Gamma; \Omega \vdash x : \tau}$$

$$\frac{\Gamma; \Omega, a : \tau_1 \vdash e : \tau_2}{\Gamma; \Omega \vdash \lambda a : \tau_1 . e : \tau_1 \multimap \tau_2} \qquad \frac{\Gamma; \Omega_1 \vdash e_1 : \tau_1 \multimap \tau_2 \qquad \Gamma; \Omega_2 \vdash e_2 : \tau_1}{\Gamma; \Omega_1 \uplus \Omega_2 \vdash e_1\ e_2 : \tau_2}$$

Case Study
Compilers: MiniML & Affi

Compiler: MiniML

Mostly identity, erases types:

$$
\begin{array}{lcl}
() & \rightsquigarrow & () \\
\mathbb{Z} & \rightsquigarrow & \mathbb{Z} \\
x & \rightsquigarrow & x \\
\lambda x : \tau.e & \rightsquigarrow & \lambda x.e^{+} \\
e_1\ e_2 & \rightsquigarrow & e_1{}^{+}\ e_2{}^{+} \\
\Lambda\alpha.e & \rightsquigarrow & \lambda\_.e^{+} \\
e{<}\tau{>} & \rightsquigarrow & e^{+}\ () \\
\texttt{ref}\ e & \rightsquigarrow & \texttt{ref}\ e^{+} \\
!e & \rightsquigarrow & !e^{+} \\
e_1 := e_2 & \rightsquigarrow & e_1{}^{+} := e_2{}^{+} \\
(\!|e|\!)_\tau & \rightsquigarrow & C_{\tau \mapsto \tau}(e^{+})
\end{array}
$$

Compiler: Affi

Wraps affine resource in thunk w/ flag to ensure single use:

$$\text{thunk}(e) \triangleq \text{let } r_{\text{fresh}} = \text{ref } \textsc{unused} \text{ in } \lambda\_.\{\text{if } !r_{\text{fresh}} \ \{\text{fail } \textsc{Conv}\} \ \{r_{\text{fresh}} := \textsc{used}; e\}\}$$

$$
\begin{array}{lcl}
() & \rightsquigarrow & () \\
x & \rightsquigarrow & x \\
a & \rightsquigarrow & a\ () \\
\lambda a : \tau.e & \rightsquigarrow & \lambda a.e^{+} \\
e_1\ e_2 & \rightsquigarrow & e_1{}^{+}\ (\text{let } x = e_2{}^{+} \text{ in } \text{thunk}(x)) \\
!e & \rightsquigarrow & e^{+} \\
\text{let } !x = e \text{ in } e' & \rightsquigarrow & \text{let } x = e^{+} \text{ in } e'^{+} \\
(e, e') & \rightsquigarrow & (e^{+}, e'^{+}) \\
\text{let } (a, a') = e \text{ in } e' & \rightsquigarrow & \text{let } x_{\text{fresh}} = e^{+} \text{ in let } a = \text{thunk}(\text{fst } x_{\text{fresh}}) \text{ in} \\
 & & \quad \text{let } a' = \text{thunk}(\text{snd } x_{\text{fresh}}) \text{ in } e'^{+} \\
(\!|e|\!)_{\tau} & \rightsquigarrow & C_{\tau \mapsto \tau}(e^{+})
\end{array}
$$

Case Study
Convertibility: MiniML & Affi

Convertibility

$$C_{\tau \mapsto \tau}, C_{\tau \mapsto \tau} : \tau \sim \tau$$

$$\frac{}{C_{\mathtt{unit} \mapsto \mathtt{unit}}, C_{\mathtt{unit} \mapsto \mathtt{unit}} : \mathtt{unit} \sim \mathtt{unit}}$$

$$\frac{C_{\tau_1 \mapsto \tau_1}, C_{\tau_1 \mapsto \tau_1} : \tau_1 \sim \tau_1 \qquad C_{\tau_2 \mapsto \tau_2}, C_{\tau_2 \mapsto \tau_2} : \tau_2 \sim \tau_2}{C_{\tau_1 \otimes \tau_2 \mapsto \tau_1 \times \tau_2}, C_{\tau_1 \times \tau_2 \mapsto \tau_1 \otimes \tau_2} : \tau_1 \otimes \tau_2 \sim \tau_1 \times \tau_2}$$

Convertibility

$$\boxed{C_{\tau \mapsto \tau}, C_{\tau \mapsto \tau} : \tau \sim \tau}$$

$$\frac{}{C_{\mathbf{unit} \mapsto \mathbf{unit}}, C_{\mathbf{unit} \mapsto \mathbf{unit}} : \mathbf{unit} \sim \mathbf{unit}}$$

$$\frac{C_{\tau_1 \mapsto \tau_1}, C_{\tau_1 \mapsto \tau_1} : \tau_1 \sim \tau_1 \qquad C_{\tau_2 \mapsto \tau_2}, C_{\tau_2 \mapsto \tau_2} : \tau_2 \sim \tau_2}{C_{\tau_1 \otimes \tau_2 \mapsto \tau_1 \times \tau_2}, C_{\tau_1 \times \tau_2 \mapsto \tau_1 \otimes \tau_2} : \tau_1 \otimes \tau_2 \sim \tau_1 \times \tau_2}$$

$$\begin{aligned}
C_{\mathbf{unit} \mapsto \mathbf{unit}}(\mathsf{e}) &\triangleq \mathsf{e} \\
C_{\mathbf{unit} \mapsto \mathbf{unit}}(\mathsf{e}) &\triangleq \mathsf{e} \\
\mathsf{C}_{\tau_1 \otimes \tau_2 \mapsto \tau_1 \times \tau_2}(\mathsf{e}) &\triangleq \mathsf{let\ x} = \mathsf{e\ in\ } (\mathsf{C}_{\tau_1 \mapsto \tau_1}(\mathsf{fst\ x}), \mathsf{C}_{\tau_1 \mapsto \tau_1}(\mathsf{snd\ x})) \\
\mathsf{C}_{\tau_1 \times \tau_2 \mapsto \tau_1 \otimes \tau_2}(\mathsf{e}) &\triangleq \mathsf{let\ x} = \mathsf{e\ in\ } (\mathsf{C}_{\tau_1 \mapsto \tau_1}(\mathsf{fst\ x}), \mathsf{C}_{\tau_1 \mapsto \tau_1}(\mathsf{snd\ x}))
\end{aligned}$$

Convertibility: functions

$$\frac{C_{\tau_1 \mapsto \tau_1}, C_{\tau_1 \mapsto \tau_1} : \tau_1 \sim \tau_1 \qquad C_{\tau_2 \mapsto \tau_2}, C_{\tau_2 \mapsto \tau_2} : \tau_2 \sim \tau_2}{C_{\tau_1 \multimap \tau_2 \mapsto (\mathtt{unit} \to \tau_1) \to \tau_2}, C_{(\mathtt{unit} \to \tau_1) \to \tau_2 \mapsto \tau_1 \multimap \tau_2} : \tau_1 \multimap \tau_2 \sim (\mathtt{unit} \to \tau_1) \to \tau_2}$$

$$\mathrm{C}_{\tau_1 \multimap \tau_2 \mapsto (\mathtt{unit} \to \tau_1) \to \tau_2}(\mathrm{e}) \triangleq$$
$$\text{let } \mathrm{x} = \mathrm{e} \text{ in } \lambda \mathrm{x}_{\mathrm{thnk}}.\text{let } \mathrm{x}_{\mathrm{conv}} = \mathrm{C}_{\tau_1 \mapsto \tau_1}(\mathrm{x}_{\mathrm{thnk}}\ ())$$
$$\text{in let } \mathrm{x}_{\mathrm{acc}} = \mathrm{thunk}(\mathrm{x}_{\mathrm{conv}}) \text{ in } \mathrm{C}_{\tau_2 \mapsto \tau_2}(\mathrm{x}\ \mathrm{x}_{\mathrm{acc}})$$

$$\mathrm{C}_{(\mathtt{unit} \to \tau_1) \to \tau_2 \mapsto \tau_1 \multimap \tau_2}(\mathrm{e}) \triangleq$$
$$\text{let } \mathrm{x} = \mathrm{e} \text{ in } \lambda \mathrm{x}_{\mathrm{thnk}}.\text{let } \mathrm{x}_{\mathrm{acc}} = \mathrm{thunk}(\mathrm{C}_{\tau_1 \mapsto \tau_1}(\mathrm{x}_{\mathrm{thnk}}\ ())) \text{ in } \mathrm{C}_{\tau_2 \mapsto \tau_2}(\mathrm{x}\ \mathrm{x}_{\mathrm{acc}})$$

Case Study
Logical Relation: MiniML & Affi

Logical Relation for MiniML & Affi

Step-indexed Kripke logical relation

$$\mathcal{V}[\![\tau]\!]_\rho = \{(W, \mathrm{v}) \mid \ldots\} \qquad \mathcal{V}[\![\tau]\!] = \{(W, \mathrm{v}) \mid \ldots\}$$

World $W = (k, \Psi, A)$ $dom(\Psi) \# dom(A)$

heap typing *flag ref cells for affine resources*

$\{\ell := \mathrm{b}, \ldots\}$

flag is b in current world

Logical Relation for MiniML & Affi

Step-indexed Kripke logical relation

$$\mathcal{V}[\![\tau]\!]_\rho = \{(W, \mathrm{v}) \mid \ldots\} \qquad \mathcal{V}[\![\tau]\!] = \{(W, \mathrm{v}) \mid \ldots\}$$

World $W = (k, \Psi, A)$ $\quad dom(\Psi)\#dom(A)$

heap typing — Ψ

flag ref cells for affine resources — A, $\{\ell := \mathrm{b}, \ldots\}$

World extension

future world — (k, Ψ', A')

$$(k, \Psi, A) \sqsubseteq (j, \Psi', A') \triangleq$$
$$j \leq k \wedge$$
$$\forall \ell \in \mathrm{dom}(\Psi).\ \Psi'(\ell) =_j \Psi(\ell) \wedge$$
$$\forall \ell \in \mathrm{dom}(A).\ell \in \mathrm{dom}(A') \wedge (A(\ell) = \mathrm{true} \implies A'(\ell) = \mathrm{true})$$

USED

Source Types as Sets of Target Terms

$$\mathcal{V}[\![\texttt{int}]\!]_\rho = \{(W, \mathsf{n}) \mid \mathsf{n} \in \mathbb{Z}\} \qquad \mathcal{V}[\![\mathbf{int}]\!]_\cdot = \{(W, \mathsf{n}) \mid \mathsf{n} \in \mathbb{Z}\}$$

$$\mathcal{V}[\![\tau_1 \times \tau_2]\!]_\rho = \{(W, (\mathsf{v}_1, \mathsf{v}_2)) \mid (W, \mathsf{v}_1) \in \mathcal{V}[\![\tau_1]\!]_\rho \wedge (W, \mathsf{v}_2) \in \mathcal{V}[\![\tau_2]\!]_\rho\}$$

$$\mathcal{V}[\![\tau_1 \to \tau_2]\!]_\rho = \{(W, \lambda \mathsf{x}.\mathsf{e}) \mid \forall \mathsf{v}, W'.\, W \sqsubset W' \wedge (W', \mathsf{v}) \in \mathcal{V}[\![\tau_1]\!]_\rho \implies (W', \mathsf{e}\{x \mapsto \mathsf{v}\}) \in \mathcal{E}[\![\tau_2]\!]_\rho\}$$

Source Types as Sets of Target Terms

$$\mathrm{thunk}(e, \ell) \triangleq \lambda\_.\mathrm{if}\ !\ell\ \{\mathrm{fail}\}\ \{\ell := \mathrm{true}; \mathrm{e}\}$$

Source Types as Sets of Target Terms

$$\text{thunk}(e, \ell) \triangleq \lambda\_.\text{if } !\ell \ \{\text{fail}\} \ \{\ell := \text{true}; \text{e}\}$$

$$\mathcal{V}[\![\tau_1 \multimap \tau_2]\!]. = \{(W, \lambda\, x.e) \mid \forall \text{v}\ W'.W \sqsubset W' \wedge (W', \text{v}) \in \mathcal{V}[\![\tau_1]\!].$$
$$W'' = (W'.k, W'.\Psi, W'.\text{A} \uplus \{\ell := \text{false}\}) \implies (W'', \text{e}\{x \mapsto \text{thunk}(v, \ell)\}) \in \mathcal{E}[\![\tau_2]\!].\}$$

Source Types as Sets of Target Terms

$$\mathrm{thunk}(e, \ell) \triangleq \lambda\_.\mathrm{if}\ !\ell\ \{\mathrm{fail}\}\ \{\ell := \mathrm{true}; \mathrm{e}\}$$

$$\mathcal{V}[\![\tau_1 \multimap \tau_2]\!]. = \{(W, \lambda\, x.e) \mid \forall \mathrm{v}\ W'.W \sqsubset W' \wedge (W', \mathrm{v}) \in \mathcal{V}[\![\tau_1]\!].$$
$$W'' = (W'.k, W'.\Psi, W'.\mathrm{A} \uplus \{\ell := \mathrm{false}\}) \implies (W'', \mathrm{e}\{x \mapsto \mathrm{thunk}(v, \ell)\}) \in \mathcal{E}[\![\tau_2]\!].\}$$

$$\mathcal{E}[\![\tau]\!]_\rho = \{(W, \mathrm{e}) \mid \forall \mathrm{H}{:}W,\ j < W.k.\ \langle \mathrm{H}, \mathrm{e}\rangle \xrightarrow{J} \langle \mathrm{H}', \mathrm{e}'\rangle \nrightarrow$$
$$\implies \mathrm{e}' = \mathrm{fail\ Conv} \vee (\exists W'.W \sqsubseteq W' \wedge \mathrm{H}' : W' \wedge (W', \mathrm{e}') \in \mathcal{V}[\![\tau]\!]_\rho)\}$$

Semantic Type Soundness: open LR

$$[\![\Delta; \Gamma \vdash e : \tau]\!] \equiv$$

$$[\![\Gamma; \Omega \vdash e : \tau]\!] \equiv$$

values substituted for affine variables must be wrapped in a thunk w/ fresh flag ref cell set to false (unused)

Recipe for Sound Lang. Interop

1. Specify convertibility rules : $\tau_{\mathtt{M}} \sim \tau_{\mathtt{A}}$
2. Implement target-level conversions for each pair of convertible types : $C_{\tau \mapsto \tau}(\cdot)$ / $C_{\tau \mapsto \tau}(\cdot)$

inhabited by target terms

To prove soundness:

3. Define semantic IR: $[\![\tau]\!] = \{e \mid \ldots\}$ $[\![\tau]\!] = \{e \mid \ldots\}$
4. Prove conversion soundness $[\![\tau \sim \tau]\!]$:
   - i. $e \in [\![\tau]\!] \implies C_{\tau \mapsto \tau}(e) \in [\![\tau]\!]$
   - ii. $e \in [\![\tau]\!] \implies C_{\tau \mapsto \tau}(e) \in [\![\tau]\!]$
5. Prove type soundness:

$[\![\Delta; \Gamma \vdash e : \tau]\!]$ and $[\![\Gamma; \Omega \vdash \mathbf{e} : \tau]\!]$

MiniML & Affi: Performant Interop.

- Sound interop. has runtime cost: all affine bindings have to be checked on use.
- An entirely Affi program would have same runtime checks even though type system *statically guarantees* they're redundant checks!

MiniML & Affi: Performant Interop.

- Sound interop. has runtime cost: all affine bindings have to be checked on use.
- An entirely Affi program would have same runtime checks even though type system *statically guarantees* they're redundant checks!
- In paper, we show how to do better:

MiniML & Affi: Performant Interop.

- Sound interop. has runtime cost: all affine bindings have to be checked on use.
- An entirely Affi program would have same runtime checks even though type system *statically guarantees* they're redundant checks!
- In paper, we show how to do better:
  - Distinguish "dynamic" Affi functions (those passed to MiniML, written $\multimap_\circ$, bind $a_\circ$) and "static" (written $\multimap_\bullet$, bind $a_\bullet$); compile "static" var/fun/app without thunks

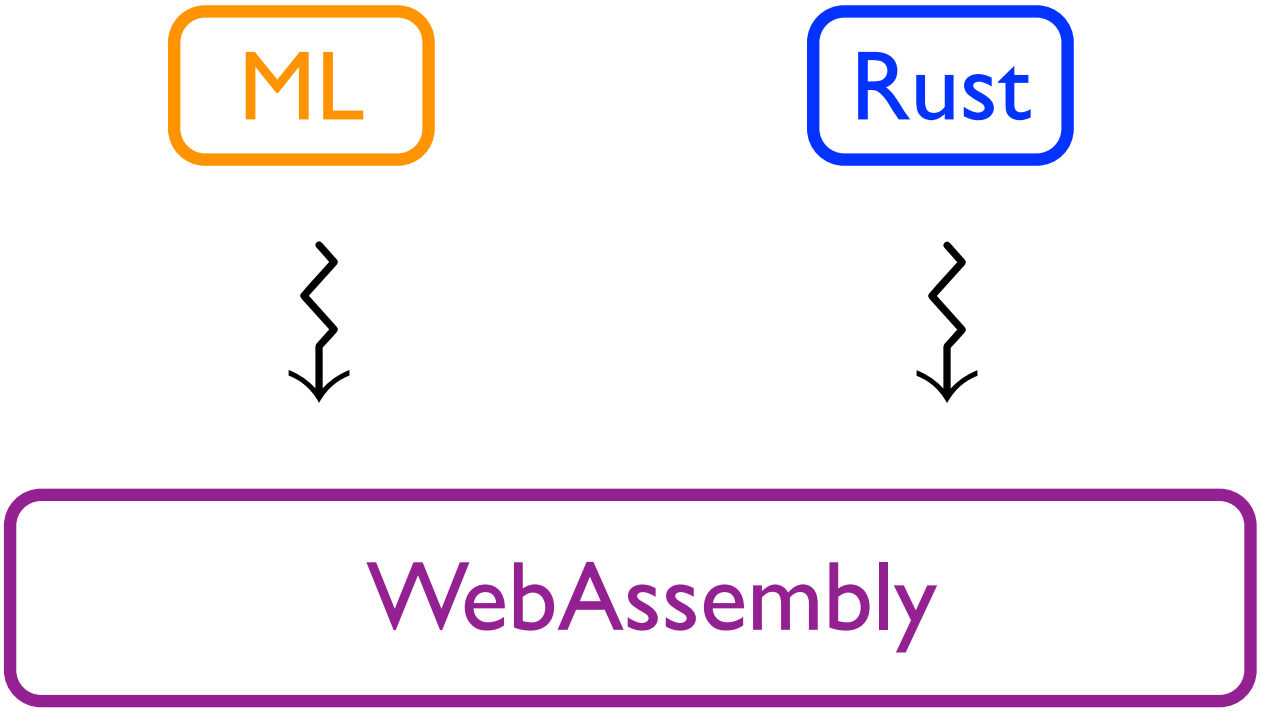
MiniML & Affi: Performant Interop.

- Sound interop. has runtime cost: all affine bindings have to be checked on use.
- An entirely Affi program would have same runtime checks even though type system *statically guarantees* they're redundant checks!
- In paper, we show how to do better:
  - Distinguish "dynamic" Affi functions (those passed to MiniML, written $\multimap_\circ$, bind $a_\circ$) and "static" (written $\multimap_\bullet$, bind $a_\bullet$); compile "static" var/fun/app without thunks
  - Need more sophisticated semantic model to ensure "static" affine variables $a_\bullet$ never used more than once

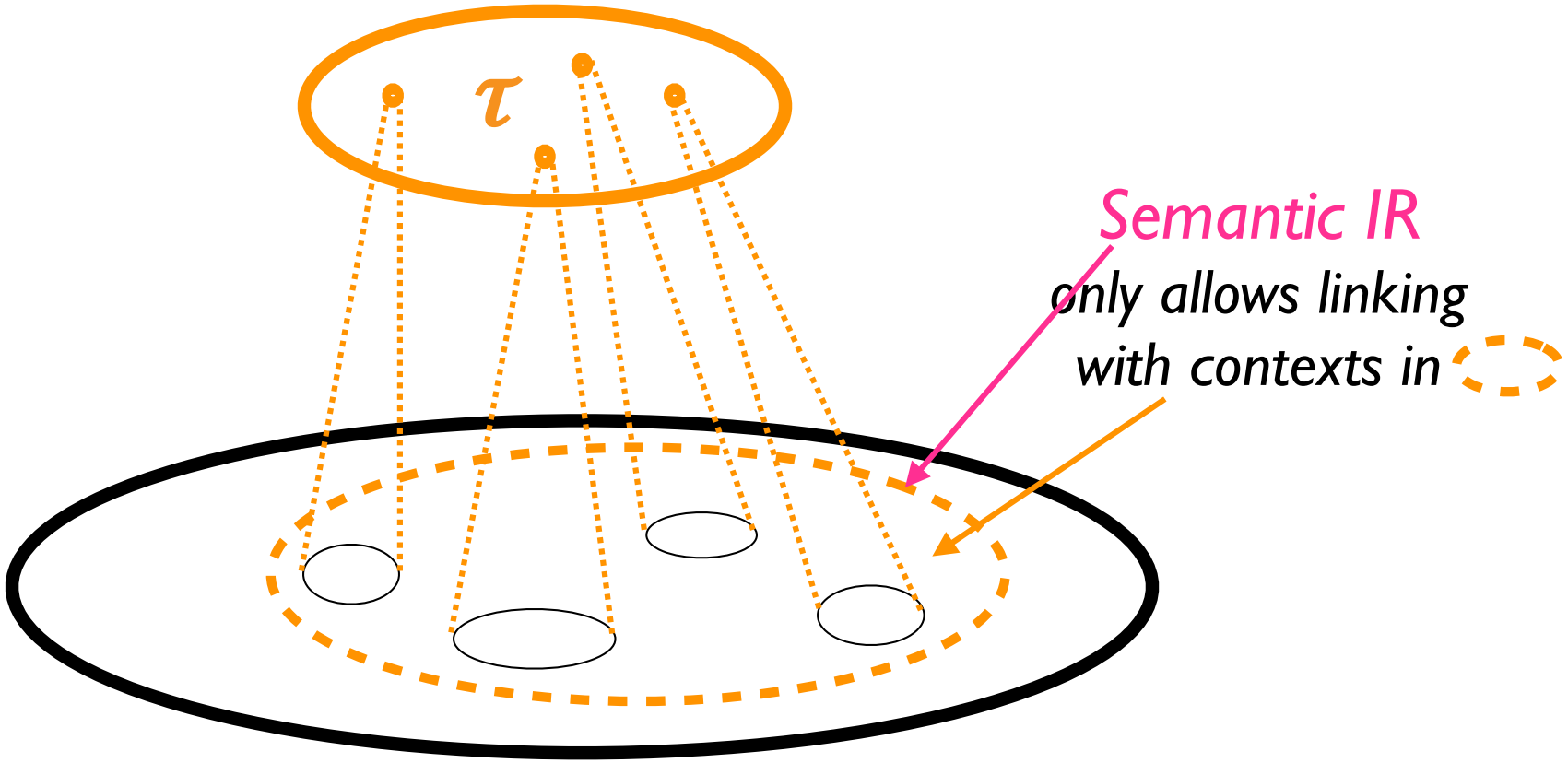
Semantic IRs

The Way Ahead

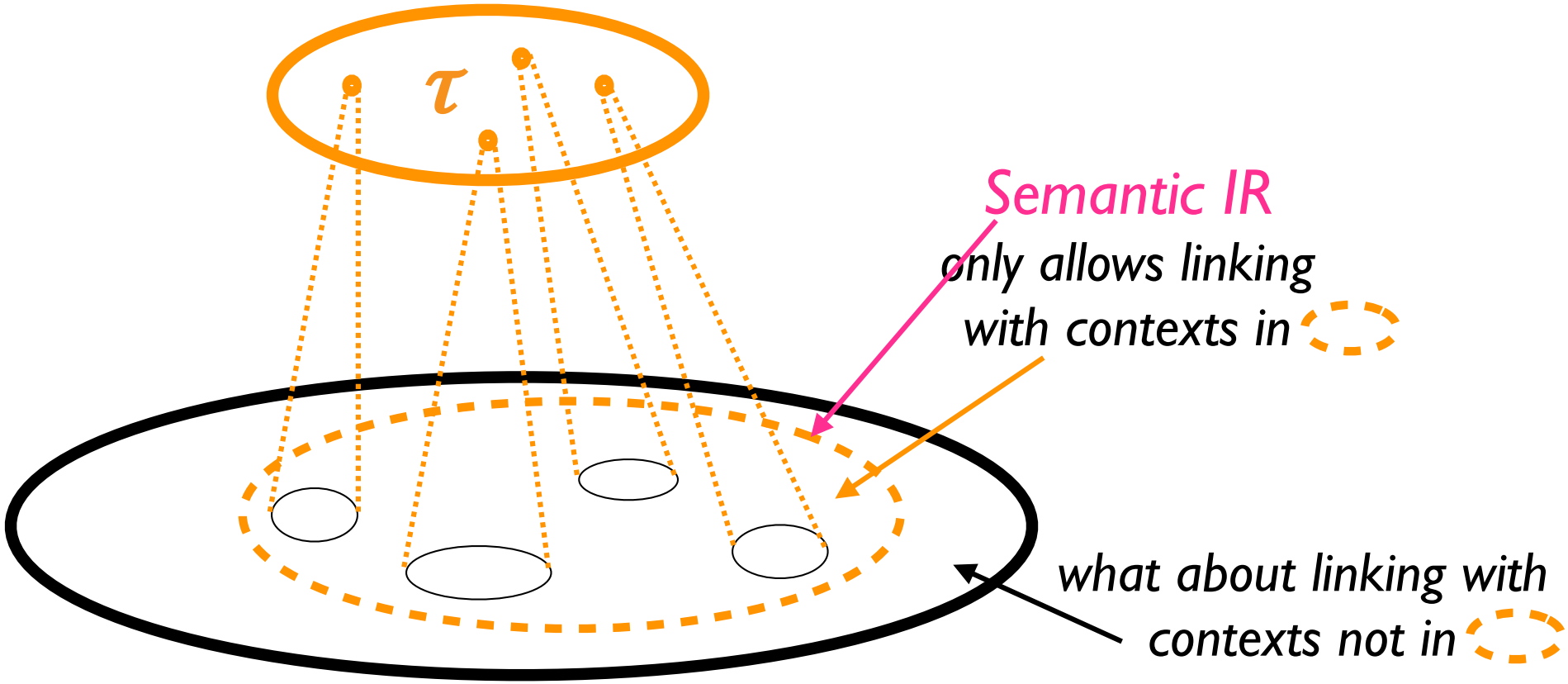
1. Future Work: Real Language Interop



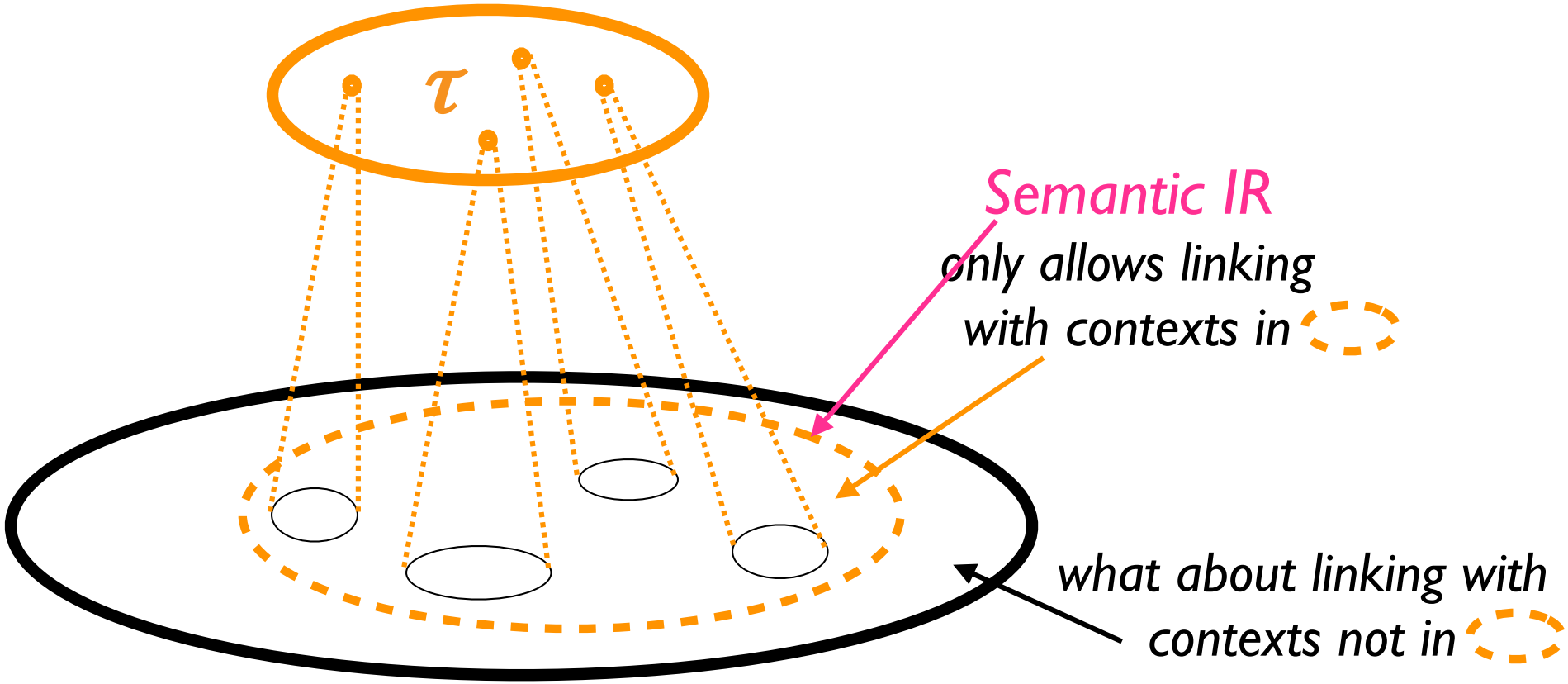
2. Open-World Soundness & Security



2. Open-World Soundness & Security



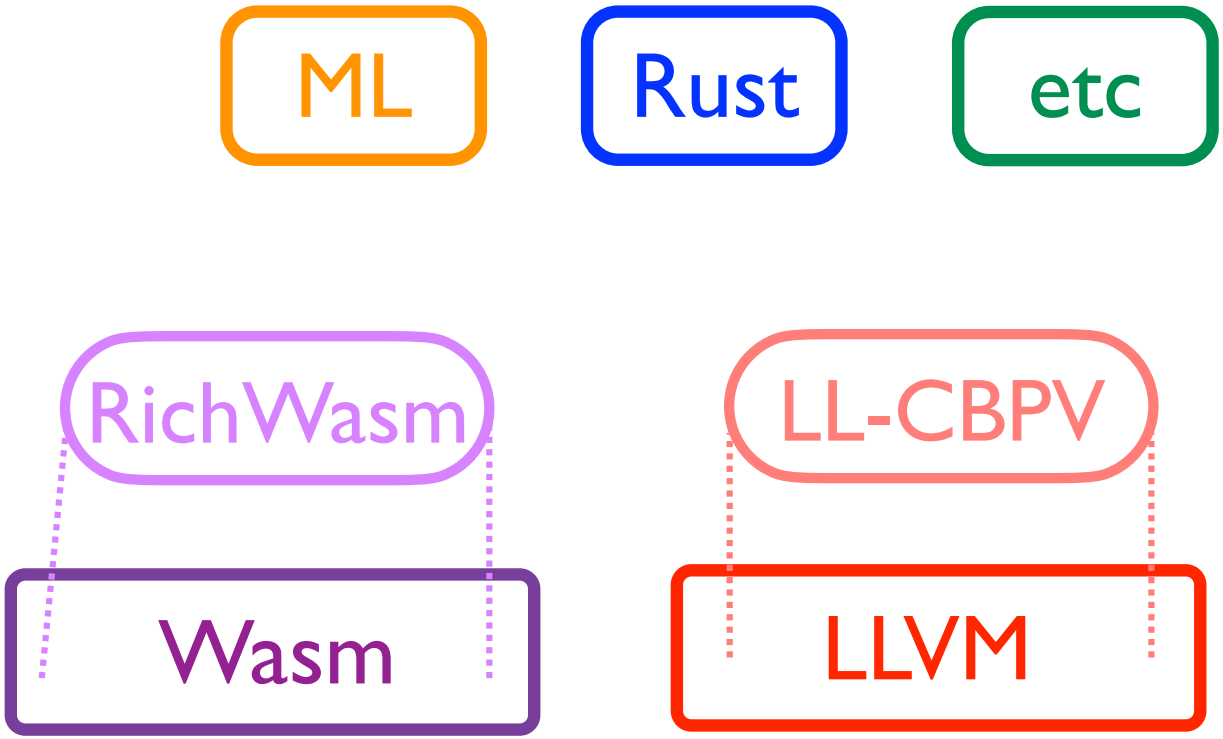
2. Open-World Soundness & Security



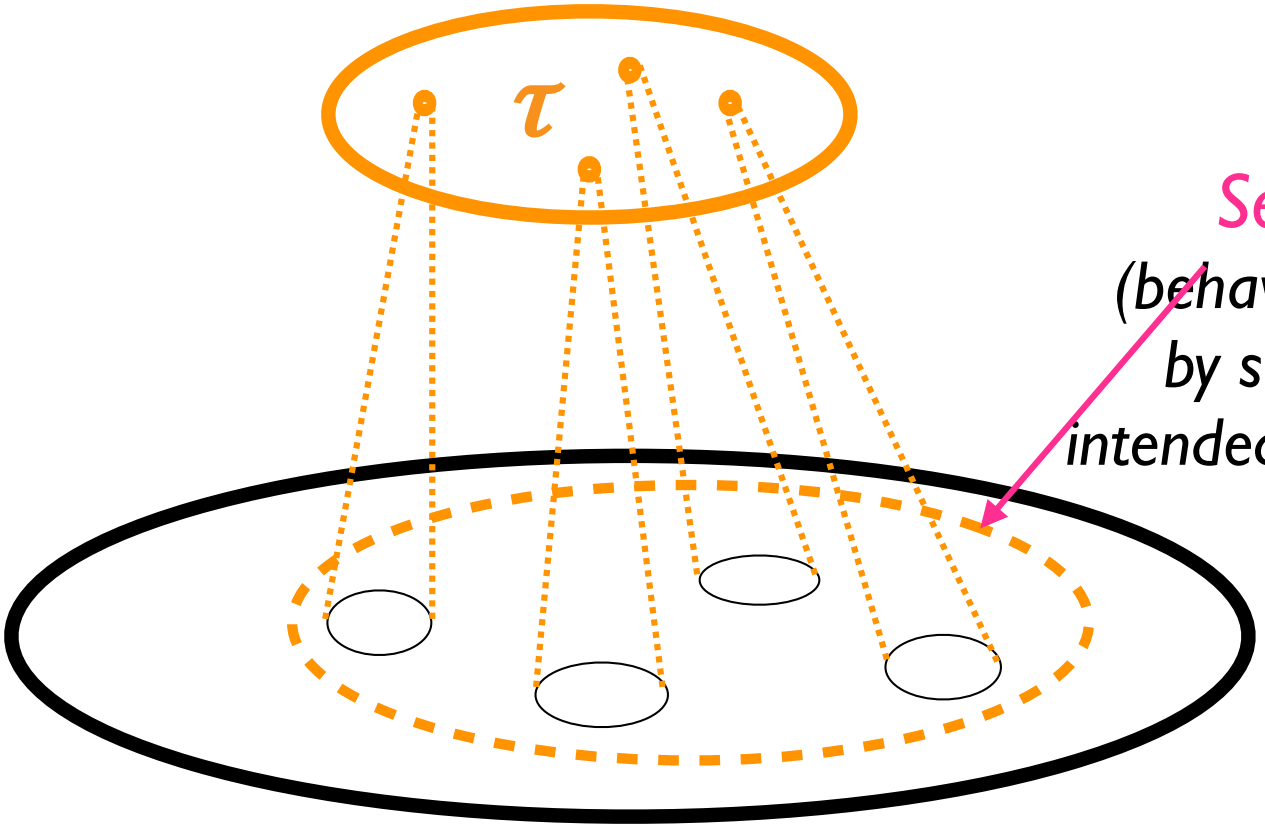
Secure compilation: show that compiled programs equivalent in can be given *protection wrappers* that ensure equivalence when linked with arbitrary contexts in

3. "Backends" for Semantic IRs

- Building semantic IRs is hard if the gap between source and target is large
- Will need to build mid-level semantic IRs that can be used to build higher-level semantic IRs



Conclusion



Semantic IR
(behave as mandated by source types in intended linking contexts)

Semantic IRs provide a foundation for designing and verifying sound (& secure) language interoperability

Questions?