Compilation as Multi-Language Semantics

William J. Bowman, University of British Columbia

Language Interoperability and Compiler Correctness

Thesis: The key problem in compiler correctness is modelling language interoperability.

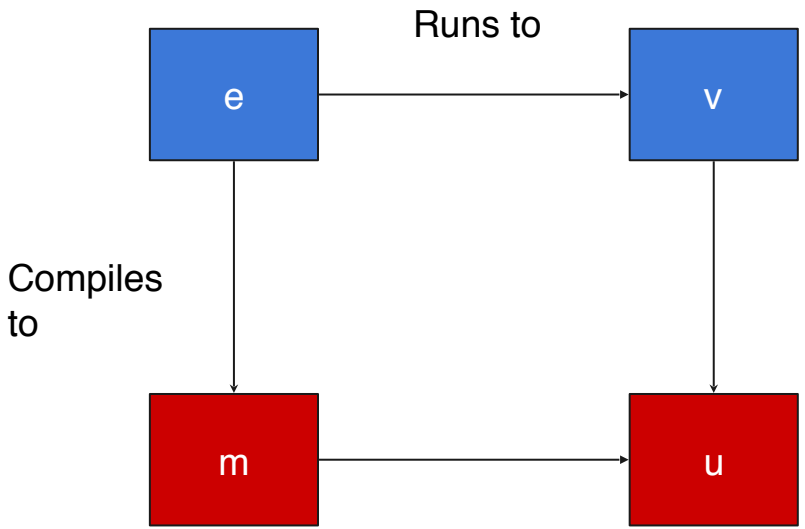
If we *start* from an interoperability semantics, we can derive a correct compiler.

In particular, model interoperability as a confluent multi-language operational semantics, and derive compilation from normalization.

When is a compiler correct, and why interoperability is key

Compiler Correctness 101

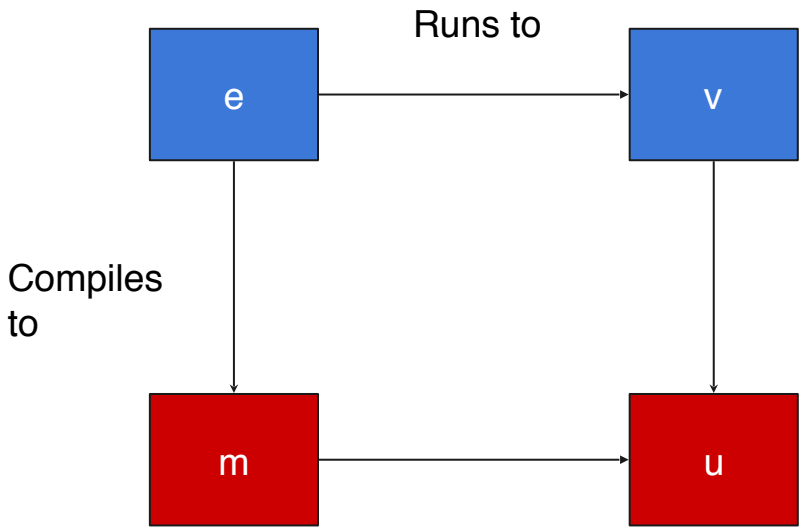
1. If e runs to v in the source
2. And[1] e translates to m
3. And[1] v translates to u
4. Then m runs to u in the target



[1] Correctness theorem could instead *guarantee*, rather than *require*, any valid term actually compiles.

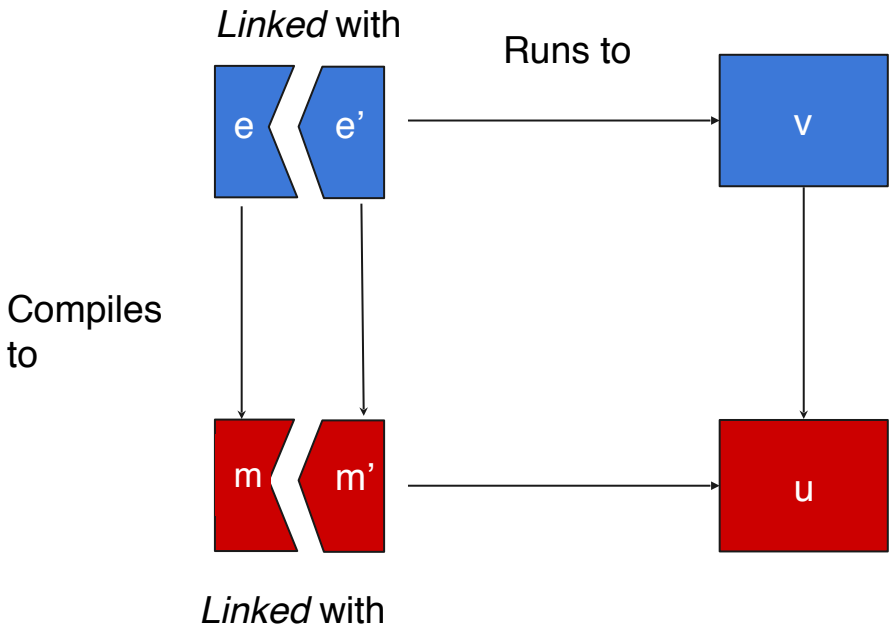
Compiler Correctness 101

This definitions only handles *whole program*: it defines correctness using evaluation.



Compiler Correctness 201:
Separate Compilation

1. If e *linked* with e' runs to v in the source
2. And[1] e translates to m
3. And[1] e' translates to m'
4. And[1] v translates to u
5. Then m *linked* with m' runs to u in the target



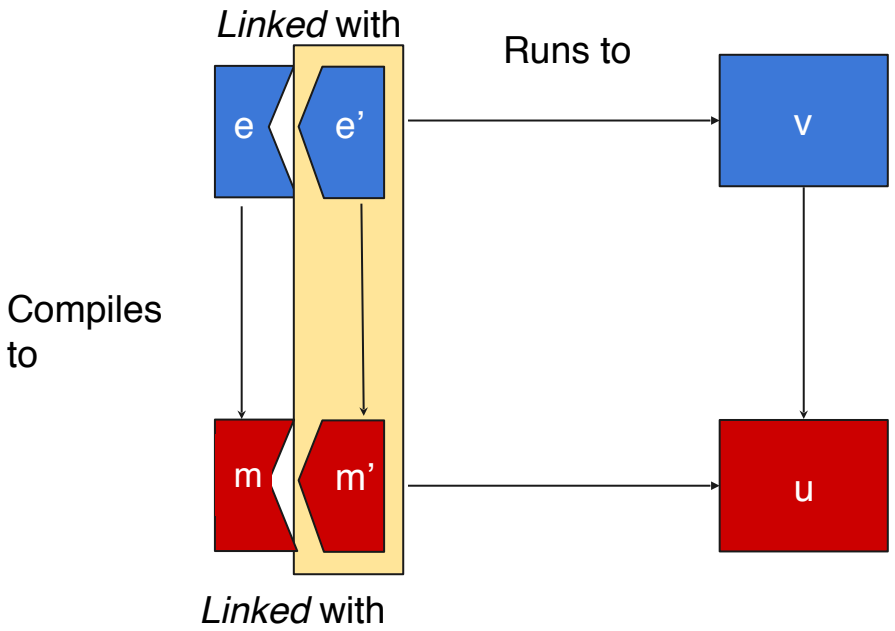
[1] Correctness theorem could instead *guarantee*, rather than *require*, any valid term actually compiles.

Compiler Correctness 201:
Separate Compilation

Can *only* link with components in the same language, compiled through the same compiler.

No guarantees with

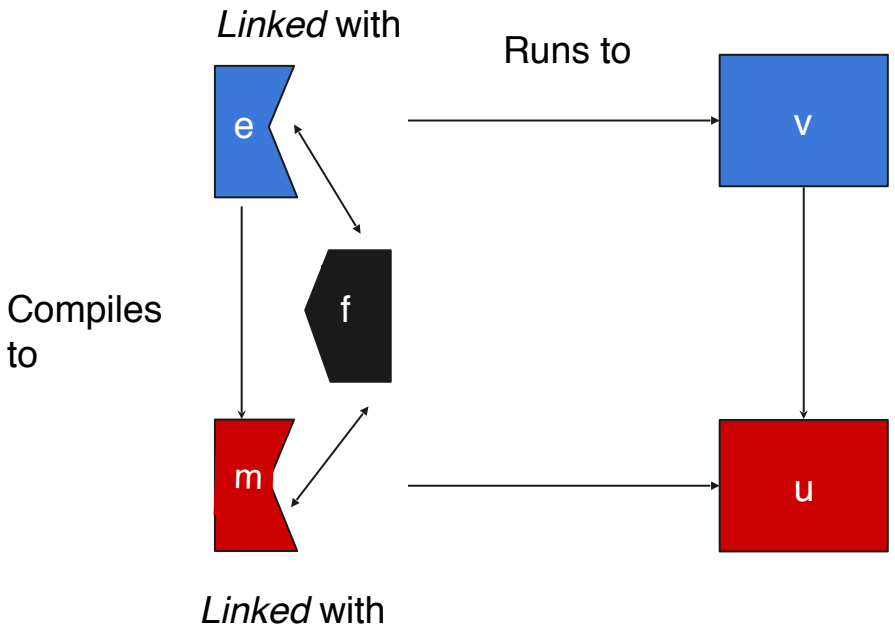
- Multiple implementations (gcc/ clang)
- Handwritten target code
- Binary-only libraries



Compiler Correctness 701:
Compositional Compilation

1. If e *linked* with **f** runs to v in the source
2. And[1] e translates to m
3. Then m *linked* with **f** runs to u in the target

Crucially, f can be a component in either the source or target[1].



[1] Or, more generally, an abstract language-independent specification of behaviour realized by a concrete implementation in (probably) the target language

Compositional Compilation is the Goal

Allows the most liberal definition of linking. No requirements on how “foreign” component **f** was produced; only that it’s implementation satisfies the spec expected by “your” component e.

Would allow us to prove a compiler correct that provide guarantees when linking with

- Output from multiple implementations (gcc/clang)
- Handwritten target code
- Binary-only libraries

How do we model linking with

- Output from multiple implementations (gcc/clang)
- Handwritten target code
- Binary-only libraries

An operational view of interoperability

Operational Semantics for Multi-Language Programs

Jacob Matthews Robert Bruce Findler
University of Chicago
{jacobm, robby}@cs.uchicago.edu

Multi-Language Operational Semantics

Suppose we have 2 languages, S and T, with operational semantics:

$e \to_s e'$

$\texttt{if false e1 e2} \to_s \texttt{e2}$

$m \to_t m'$

$\texttt{if 0 m1 m2} \to_t \texttt{m2}$

Multi-Language Operational Semantics

Suppose we have 2 languages, S and T, with operational semantics:

$$\frac{e \to_s e'}{\texttt{if false e1 e2} \to_s \texttt{e2}} \qquad \frac{m \to_t m'}{\texttt{if 0 m1 m2} \to_t \texttt{m2}}$$

Multi-Language Operational Semantics

Suppose we have 2 languages, S and T, with operational semantics:

And we want them to be able to interoperate arbitrarily.

$e \to_s e'$

$\texttt{if false e1 e2} \to_s \texttt{e2}$

$m \to_t m'$

$\texttt{if 0 m1 m2} \to_t \texttt{m2}$

p

$p ::= e \mid m$

$\texttt{if 0 e1 e2} \in p$

Multi-Language Operational Semantics

Suppose we have 2 languages, S and T, with operational semantics:

And we want them to be able to interoperate arbitrarily.

Suffices to define how to move between languages “at the boundaries”, and otherwise reuse the other semantics.

$e \to_s e'$

$m \to_t m'$

$\texttt{if false e1 e2} \to_s \texttt{e2}$

$\texttt{if 0 m1 m2} \to_t \texttt{m2}$

p

$p ::= e \mid m$

$\texttt{if 0 e1 e2} \in p$

$m ::= \ldots. \mid TS(e)$

$e ::= \ldots. \mid ST(m)$

$\texttt{if ST(0) e1 e2} \in p, e$

Multi-Language Operational Semantics

Suppose we have 2 languages, S and T, with operational semantics:

And we want them to be able to interoperate arbitrarily.

Suffices to define how to move between languages “at the boundaries”, and otherwise reuse the other semantics.

$e \to_s e'$

$m \to_t m'$

$\texttt{if false e1 e2} \to_s \texttt{e2}$

$\texttt{if 0 m1 m2} \to_t \texttt{m2}$

p

$p ::= e \mid m$

$\texttt{if 0 e1 e2} \in p$

$m ::= \ldots. \mid TS(e)$

$e ::= \ldots. \mid ST(m)$

$\texttt{if ST(0) e1 e2} \in p, e$

$p \to p'$

$$\frac{p \to_s p}{p \to p'} \qquad \frac{p \to_t p}{p \to p'}$$

Multi-Language Operational Semantics

Suppose we have 2 languages, S and T, with operational semantics:

And we want them to be able to interoperate arbitrarily.

Suffices to define how to move between languages “at the boundaries”, and otherwise reuse the other semantics.

$e \to_s e'$

$m \to_t m'$

$\texttt{if false e1 e2} \to_s \texttt{e2}$

$\texttt{if 0 m1 m2} \to_t \texttt{m2}$

p

$p ::= e \mid m$

$\texttt{if 0 e1 e2} \in p$

$m ::= \ldots. \mid \texttt{TS}(e)$

$e ::= \ldots. \mid \texttt{ST}(m)$

$\texttt{if ST(0) e1 e2} \in p, e$

$p \to p'$

$\texttt{TS(false)} \to \texttt{0}$

$\texttt{ST(0)} \to \texttt{false}$

$$\frac{p \to_s p}{p \to p'} \qquad \frac{p \to_t p}{p \to p'}$$

Multi-Language Operational Semantics

Suppose we have 2 languages, S and T, with operational semantics:

And we want them to be able to interoperate arbitrarily.

Suffices to define how to move between languages “at the boundaries”, and otherwise reuse the other semantics.

$e \to_s e'$

$m \to_t m'$

$\texttt{if false e1 e2} \to_s \texttt{e2}$

$\texttt{if 0 m1 m2} \to_t \texttt{m2}$

p

$p ::= e \mid m$

$\texttt{if 0 e1 e2} \in p$

$m ::= \ldots. \mid \texttt{TS}(e)$

$e ::= \ldots. \mid \texttt{ST}(m)$

$\texttt{if ST(0) e1 e2} \in p, e$

$p \to p'$

$\texttt{TS(false)} \to \texttt{0}$

$\texttt{ST(0)} \to \texttt{false}$

$$\frac{p \to_s p}{p \to p'} \qquad \frac{p \to_t p}{p \to p'}$$

Observation:
What is this if not a compiler?

Multi-Language Semantics in Compilation

Compiler:

$$\frac{\begin{array}{c} y_1, \ldots, y_m = \mathrm{fv}(\lambda[\overline{\alpha}](\overline{x:\tau}).t) \qquad \beta_1, \ldots, \beta_k = \mathrm{ftv}(\lambda[\overline{\alpha}](\overline{x:\tau}).t) \\ \Delta, \overline{\alpha}; \Gamma, \overline{x:\tau} \vdash t : \tau' \rightsquigarrow \mathbf{t} \qquad \tau_{\mathbf{env}} = \langle (\Gamma(y_1))^{\mathcal{C}}, \ldots, (\Gamma(y_m))^{\mathcal{C}} \rangle \\ \mathbf{v} = \boldsymbol{\lambda}[\overline{\beta}, \overline{\alpha}](\mathbf{z} : \tau_{\mathbf{env}}, \overline{\mathbf{x} : \tau^{\mathcal{C}}}).(\mathbf{t}[\pi_1(\mathbf{z})/\mathbf{y_1}] \cdots [\pi_m(\mathbf{z})/\mathbf{y_m}]) \end{array}}{\begin{array}{l} \Delta; \Gamma \vdash \lambda[\overline{\alpha}](\overline{x:\tau}).t : \forall[\overline{\alpha}].(\overline{\tau}) \to \tau' \rightsquigarrow \\ \qquad \mathbf{pack}\langle \tau_{\mathbf{env}}, \langle \mathbf{v}[\overline{\beta}], \langle \overline{\mathbf{y}} \rangle \rangle \rangle \ \mathbf{as} \ \exists \alpha'.\langle (\forall[\overline{\alpha}].(\alpha', \overline{\tau}^{\mathcal{C}}) \to \tau'^{\mathcal{C}}), \alpha' \rangle \end{array}}$$

Interoperability Semantics:

$$\mathbf{CF}^{\forall[\overline{\alpha}].(\overline{\tau}) \to \tau'}(v) = \mathbf{pack}\langle \mathbf{unit}, \langle \mathbf{v}, () \rangle \rangle \ \mathbf{as} \ (\forall[\overline{\alpha}].(\overline{\tau}) \to \tau')^{\langle \mathcal{C} \rangle}$$

$$\text{where } \mathbf{v} = \boldsymbol{\lambda}[\overline{\alpha}](\mathbf{z} : \mathbf{unit}, \overline{\mathbf{x} : \tau^{\langle \mathcal{C} \rangle}[\alpha/\lceil\alpha\rceil]}).\mathcal{C}\mathcal{F}^{\tau'[\mathsf{L}\langle\alpha\rangle/\alpha]}(\mathbf{v}\ [\overline{\mathsf{L}\langle\alpha\rangle}]\ \overline{\tau[\mathsf{L}\langle\alpha\rangle/\alpha]\mathcal{F}\mathcal{C}\mathbf{x}})$$

Perconti, J. T. & Ahmed, A.
Verifying an Open Compiler Using Multi-language Semantics *(ESOP)*, **2014**

Multi-Language Semantics in Compilation

Compiler:

$$\frac{\begin{array}{c} y_1, \ldots, y_m = \mathrm{fv}(\lambda[\overline{\alpha}](\overline{x:\tau}).t) \qquad \beta_1, \ldots, \beta_k = \mathrm{ftv}(\lambda[\overline{\alpha}](\overline{x:\tau}).t) \\ \Delta, \overline{\alpha}; \Gamma, \overline{x:\tau} \vdash t : \tau' \rightsquigarrow t \qquad \tau_{env} = \langle (\Gamma(y_1))^{\mathcal{C}}, \ldots, (\Gamma(y_m))^{\mathcal{C}} \rangle \\ v = \lambda[\overline{\beta}, \overline{\alpha}](z : \tau_{env}, \overline{x : \tau^{\mathcal{C}}}).(t[\pi_1(z)/y_1] \cdots [\pi_m(z)/y_m]) \end{array}}{\Delta; \Gamma \vdash \lambda[\overline{\alpha}](\overline{x:\tau}).t : \forall[\overline{\alpha}].(\overline{\tau}) \to \tau' \rightsquigarrow \mathbf{pack}\langle \tau_{env}, \langle v[\overline{\beta}], \langle \overline{y} \rangle \rangle \rangle \text{ as } \exists \alpha'.\langle (\forall[\overline{\alpha}].(\alpha', \overline{\tau}^{\mathcal{C}}) \to \tau'^{\mathcal{C}}), \alpha' \rangle}$$

Interoperability Semantics:

$$\mathbf{CF}^{\forall[\overline{\alpha}].(\overline{\tau}) \to \tau'}(v) = \mathbf{pack}\langle \mathbf{unit}, \langle v, () \rangle \rangle \text{ as } (\forall[\overline{\alpha}].(\overline{\tau}) \to \tau')^{\langle \mathcal{C} \rangle}$$

$$\text{where } v = \lambda[\overline{\alpha}](z : \mathbf{unit}, \overline{x : \tau^{\langle \mathcal{C} \rangle}[\alpha/\lceil \alpha \rceil]}).\mathcal{C}\mathcal{F}^{\tau'[\mathsf{L}\langle\alpha\rangle/\alpha]}(v\ [\mathsf{L}\langle\alpha\rangle]\ \overline{\tau[\mathsf{L}\langle\alpha\rangle/\alpha]\mathcal{F}\mathcal{C}x})$$

Perconti, J. T. & Ahmed, A.
Verifying an Open Compiler Using Multi-language Semantics *(ESOP)*, **2014**

Multi-Language Semantics in Compilation

Compiler:

$$\frac{\begin{array}{c} y_1, \ldots, y_m = \mathrm{fv}(\lambda[\overline{\alpha}](\overline{x:\tau}).t) \qquad \beta_1, \ldots, \beta_k = \mathrm{ftv}(\lambda[\overline{\alpha}](\overline{x:\tau}).t) \\ \Delta, \overline{\alpha}; \Gamma, \overline{x:\tau} \vdash t : \tau' \rightsquigarrow t \qquad \tau_{env} = \langle (\Gamma(y_1))^{\mathcal{C}}, \ldots, (\Gamma(y_m))^{\mathcal{C}} \rangle \\ v = \lambda[\overline{\beta}, \overline{\alpha}](z : \tau_{env}, \overline{x : \tau^{\mathcal{C}}}).(t[\pi_1(z)/y_1] \cdots [\pi_m(z)/y_m]) \end{array}}{\Delta; \Gamma \vdash \lambda[\overline{\alpha}](\overline{x:\tau}).t : \forall[\overline{\alpha}].(\overline{\tau}) \to \tau' \rightsquigarrow \mathbf{pack}\langle \tau_{env}, \langle v[\overline{\beta}], \langle \overline{y} \rangle \rangle \rangle \text{ as } \exists \alpha'.\langle (\forall[\overline{\alpha}].(\alpha', \overline{\tau}^{\mathcal{C}}) \to \tau'^{\mathcal{C}}), \alpha' \rangle}$$

Interoperability Semantics:

$$\mathrm{CF}^{\forall[\overline{\alpha}].(\overline{\tau}) \to \tau'}(v) = \mathbf{pack}\langle \mathbf{unit}, \langle v, () \rangle \rangle \text{ as } (\forall[\overline{\alpha}].(\overline{\tau}) \to \tau')^{\langle \mathcal{C} \rangle}$$

$$\text{where } v = \lambda[\overline{\alpha}](z : \mathbf{unit}, \overline{x : \tau^{\langle \mathcal{C} \rangle}[\alpha/\lceil \alpha \rceil]}).\mathcal{C}\mathcal{F}^{\tau'[\mathsf{L}\langle \alpha \rangle/\alpha]}(v\ [\mathsf{L}\langle \alpha \rangle]\ \overline{\tau[\mathsf{L}\langle \alpha \rangle/\alpha]\mathcal{F}\mathcal{C}x})$$

Perconti, J. T. & Ahmed, A.
Verifying an Open Compiler Using Multi-language Semantics *(ESOP)*, **2014**

Multi-Language Semantics in Compilation

Compiler:

$$\frac{\mathsf{y}_1, \ldots, \mathsf{y}_m = \mathrm{fv}(\lambda[\overline{\alpha}](\overline{\mathsf{x}:\tau}).\mathsf{t}) \qquad \beta_1, \ldots, \beta_k = \mathrm{ftv}(\lambda[\overline{\alpha}](\overline{\mathsf{x}:\tau}).\mathsf{t}) \\ \Delta, \overline{\alpha}; \Gamma, \overline{\mathsf{x}:\tau} \vdash \mathsf{t}:\tau' \rightsquigarrow \mathbf{t} \qquad \tau_{\mathbf{env}} = \langle (\Gamma(\mathsf{y}_1))^{\mathcal{C}}, \ldots, (\Gamma(\mathsf{y}_m))^{\mathcal{C}} \rangle \\ \mathbf{v} = \lambda[\overline{\beta}, \overline{\alpha}](\mathbf{z}:\tau_{\mathbf{env}}, \overline{\mathbf{x}:\tau^{\mathcal{C}}}).(\mathbf{t}[\pi_1(\mathbf{z})/\mathbf{y}_1] \cdots [\pi_m(\mathbf{z})/\mathbf{y}_m])}{\Delta; \Gamma \vdash \lambda[\overline{\alpha}](\overline{\mathsf{x}:\tau}).\mathsf{t}:\forall[\overline{\alpha}].(\overline{\tau}) \to \tau' \rightsquigarrow \mathbf{pack}\langle \tau_{\mathbf{env}}, \langle \mathbf{v}[\overline{\beta}], \langle \overline{\mathbf{y}} \rangle \rangle \rangle \mathbf{as} \exists \alpha'.\langle (\forall[\overline{\alpha}].(\alpha', \overline{\tau}^{\mathcal{C}}) \to \tau'^{\mathcal{C}}), \alpha' \rangle}$$

Interoperability Semantics:

$$\mathrm{CF}^{\forall[\overline{\alpha}].(\overline{\tau}) \to \tau'}(\mathsf{v}) = \mathbf{pack}\langle \mathbf{unit}, \langle \mathsf{v}, () \rangle \rangle \mathbf{as} (\forall[\overline{\alpha}].(\overline{\tau}) \to \tau')^{\langle \mathcal{C} \rangle}$$

$$\text{where } \mathbf{v} = \lambda[\overline{\alpha}](\mathbf{z}:\mathbf{unit}, \overline{\mathbf{x}:\tau^{\langle \mathcal{C} \rangle}[\alpha/\lceil\alpha\rceil]}).\mathcal{C}\mathcal{F}^{\tau'[\mathsf{L}\langle\alpha\rangle/\alpha]}(\mathsf{v}\,[\mathsf{L}\langle\alpha\rangle]\ \overline{\tau[\mathsf{L}\langle\alpha\rangle/\alpha]\mathcal{F}\mathcal{C}\mathbf{x}})$$

Perconti, J. T. & Ahmed, A.
Verifying an Open Compiler Using Multi-language Semantics *(ESOP)*, **2014**

Multi-Language Semantics in Compilation

Compiler:

$$\frac{\begin{array}{l}y_1, \ldots, y_m = \mathrm{fv}(\lambda[\overline{\alpha}](\overline{x:\tau}).t) \qquad \beta_1, \ldots, \beta_k = \mathrm{ftv}(\lambda[\overline{\alpha}](\overline{x:\tau}).t) \\ \Delta, \overline{\alpha}; \Gamma, \overline{x:\tau} \vdash t : \tau' \rightsquigarrow t \qquad \tau_{env} = \langle (\Gamma(y_1))^{\mathcal{C}}, \ldots, (\Gamma(y_m))^{\mathcal{C}} \rangle \\ v = \lambda[\overline{\beta}, \overline{\alpha}](z : \tau_{env}, \overline{x : \tau^{\mathcal{C}}}).(t[\pi_1(z)/y_1] \cdots [\pi_m(z)/y_m])\end{array}}{\Delta; \Gamma \vdash \lambda[\overline{\alpha}](\overline{x:\tau}).t : \forall[\overline{\alpha}].(\overline{\tau}) \to \tau' \rightsquigarrow \mathbf{pack}\langle \tau_{env}, \langle v[\overline{\beta}], \langle \overline{y} \rangle \rangle \rangle \text{ as } \exists \alpha'.\langle (\forall[\overline{\alpha}].(\alpha', \overline{\tau}^{\mathcal{C}}) \to \tau'^{\mathcal{C}}), \alpha' \rangle}$$

Interoperability Semantics:

$$\mathrm{CF}^{\forall[\overline{\alpha}].(\overline{\tau}) \to \tau'}(v) = \mathbf{pack}\langle \mathbf{unit}, \langle v, () \rangle \rangle \text{ as } (\forall[\overline{\alpha}].(\overline{\tau}) \to \tau')^{\langle \mathcal{C} \rangle}$$

$$\text{where } v = \lambda[\overline{\alpha}](z : \mathbf{unit}, \overline{x : \tau^{\langle \mathcal{C} \rangle}[\alpha/\lceil \alpha \rceil]}).\mathcal{C}\mathcal{F}^{\tau'[\overline{\mathsf{L}\langle\alpha\rangle/\alpha}]}(v\ [\overline{\mathsf{L}\langle\alpha\rangle}]\ \overline{\tau[\mathsf{L}\langle\alpha\rangle/\alpha]\mathcal{F}\mathcal{C}x})$$

Perconti, J. T. & Ahmed, A.
Verifying an Open Compiler Using Multi-language Semantics *(ESOP)*, **2014**

Multi-Language Semantics in Compilation

Compiler:

$$\frac{\begin{array}{l} y_1, \ldots, y_m = \mathrm{fv}(\lambda[\overline{\alpha}](\overline{x:\tau}).t) \qquad \beta_1, \ldots, \beta_k = \mathrm{ftv}(\lambda[\overline{\alpha}](\overline{x:\tau}).t) \\ \Delta, \overline{\alpha}; \Gamma, \overline{x:\tau} \vdash t : \tau' \rightsquigarrow t \qquad \tau_{env} = \langle (\Gamma(y_1))^{\mathcal{C}}, \ldots, (\Gamma(y_m))^{\mathcal{C}} \rangle \\ v = \lambda[\overline{\beta}, \overline{\alpha}](z : \tau_{env}, \overline{x : \tau^{\mathcal{C}}}).(t[\pi_1(z)/y_1] \cdots [\pi_m(z)/y_m]) \end{array}}{\Delta; \Gamma \vdash \lambda[\overline{\alpha}](\overline{x:\tau}).t : \forall[\overline{\alpha}].(\overline{\tau}) \to \tau' \rightsquigarrow \mathbf{pack}\langle \tau_{env}, \langle v[\overline{\beta}], \langle \overline{y} \rangle \rangle \rangle \mathbf{as} \exists \alpha'.\langle (\forall[\overline{\alpha}].(\alpha', \overline{\tau}^{\mathcal{C}}) \to \tau'^{\mathcal{C}}), \alpha' \rangle}$$

Interoperability Semantics:

$$\mathrm{CF}^{\forall[\overline{\alpha}].(\overline{\tau}) \to \tau'}(v) = \mathbf{pack}\langle \mathbf{unit}, \langle v, () \rangle \rangle \mathbf{as} \, (\forall[\overline{\alpha}].(\overline{\tau}) \to \tau')^{\langle \mathcal{C} \rangle}$$

$$\text{where } v = \lambda[\overline{\alpha}](z : \mathbf{unit}, \overline{x : \tau^{\langle \mathcal{C} \rangle}[\alpha/\lceil \alpha \rceil]}).\mathcal{C}\mathcal{F}^{\tau'[\overline{\mathsf{L}\langle\alpha\rangle/\alpha}]}(v\,[\overline{\mathsf{L}\langle\alpha\rangle}]\,\overline{\tau[\mathsf{L}\langle\alpha\rangle/\alpha]\mathcal{F}\mathcal{C}x})$$

Perconti, J. T. & Ahmed, A.
Verifying an Open Compiler Using Multi-language Semantics *(ESOP)*, **2014**

Multi-Language Semantics in Compilation

Compiler:

$$\frac{\begin{array}{c} y_1, \ldots, y_m = \mathrm{fv}(\lambda[\overline{\alpha}](\overline{x:\tau}).t) \qquad \beta_1, \ldots, \beta_k = \mathrm{ftv}(\lambda[\overline{\alpha}](\overline{x:\tau}).t) \\ \Delta, \overline{\alpha}; \Gamma, \overline{x:\tau} \vdash t : \tau' \rightsquigarrow t \qquad \tau_{env} = \langle (\Gamma(y_1))^{\mathcal{C}}, \ldots, (\Gamma(y_m))^{\mathcal{C}} \rangle \\ v = \lambda[\overline{\beta}, \overline{\alpha}](z : \tau_{env}, \overline{x : \tau^{\mathcal{C}}}).(t[\pi_1(z)/y_1] \cdots [\pi_m(z)/y_m]) \end{array}}{\Delta; \Gamma \vdash \lambda[\overline{\alpha}](\overline{x:\tau}).t : \forall[\overline{\alpha}].(\overline{\tau}) \to \tau' \rightsquigarrow \mathbf{pack}\langle \tau_{env}, \langle v[\overline{\beta}], \langle \overline{y} \rangle \rangle \rangle \text{ as } \exists \alpha'.\langle (\forall[\overline{\alpha}].(\alpha', \overline{\tau}^{\mathcal{C}}) \to \tau'^{\mathcal{C}}), \alpha' \rangle}$$

Interoperability Semantics:

$$\mathrm{CF}^{\forall[\overline{\alpha}].(\overline{\tau}) \to \tau'}(v) = \mathbf{pack}\langle \mathbf{unit}, \langle v, () \rangle \rangle \text{ as } (\forall[\overline{\alpha}].(\overline{\tau}) \to \tau')^{\langle \mathcal{C} \rangle}$$

$$\text{where } v = \lambda[\overline{\alpha}](z : \mathbf{unit}, \overline{x : \tau^{\langle \mathcal{C} \rangle}[\alpha/\lceil \alpha \rceil]}).\mathcal{C}\mathcal{F}^{\tau'[\overline{\mathsf{L}\langle\alpha\rangle/\alpha}]}(v\ [\overline{\mathsf{L}\langle\alpha\rangle}]\ \overline{\tau[\mathsf{L}\langle\alpha\rangle/\alpha]\mathcal{F}\mathcal{C}x})$$

Perconti, J. T. & Ahmed, A.
Verifying an Open Compiler Using Multi-language Semantics *(ESOP)*, **2014**

Multi-Language Semantics in Compilation

Compiler:

$$\frac{\begin{array}{c} y_1, \ldots, y_m = \mathrm{fv}(\lambda[\overline{\alpha}](\overline{x:\tau}).t) \qquad \beta_1, \ldots, \beta_k = \mathrm{ftv}(\lambda[\overline{\alpha}](\overline{x:\tau}).t) \\ \Delta, \overline{\alpha}; \Gamma, \overline{x:\tau} \vdash t : \tau' \rightsquigarrow t \qquad \tau_{env} = \langle (\Gamma(y_1))^{\mathcal{C}}, \ldots, (\Gamma(y_m))^{\mathcal{C}} \rangle \\ v = \lambda[\overline{\beta}, \overline{\alpha}](z : \tau_{env}, \overline{x : \tau^{\mathcal{C}}}).(t[\pi_1(z)/y_1] \cdots [\pi_m(z)/y_m]) \end{array}}{\begin{array}{l} \Delta; \Gamma \vdash \lambda[\overline{\alpha}](\overline{x:\tau}).t : \forall[\overline{\alpha}].(\overline{\tau}) \to \tau' \rightsquigarrow \\ \qquad \mathbf{pack}\langle \tau_{env}, \langle v[\overline{\beta}], \langle \overline{y} \rangle \rangle \rangle \ \mathbf{as}\ \exists \alpha'.\langle (\forall[\overline{\alpha}].(\alpha', \overline{\tau}^{\mathcal{C}}) \to \tau'^{\mathcal{C}}), \alpha' \rangle \end{array}}$$

Interoperability Semantics:

$$\mathbf{CF}^{\forall[\overline{\alpha}].(\overline{\tau}) \to \tau'}(v) = \mathbf{pack}\langle \mathbf{unit}, \langle v, () \rangle \rangle \ \mathbf{as}\ (\forall[\overline{\alpha}].(\overline{\tau}) \to \tau')^{\langle \mathcal{C} \rangle}$$

$$\text{where } v = \lambda[\overline{\alpha}](z : \mathbf{unit}, \overline{x : \tau^{\langle \mathcal{C} \rangle}[\alpha/\lceil \alpha \rceil]}).\mathcal{C}\mathcal{F}^{\tau'[\overline{L\langle \alpha \rangle/\alpha}]}(v\ [\overline{L\langle \alpha \rangle}]\ \overline{\tau[L\langle \alpha \rangle/\alpha]}\mathcal{F}\mathcal{C}x)$$

Perconti, J. T. & Ahmed, A.
Verifying an Open Compiler Using Multi-language Semantics *(ESOP)*, **2014**

Multi-Language Semantics in Compilation

Compiler:

$$\frac{\begin{array}{c} y_1, \ldots, y_m = \mathrm{fv}(\lambda[\overline{\alpha}](\overline{x:\tau}).t) \qquad \beta_1, \ldots, \beta_k = \mathrm{ftv}(\lambda[\overline{\alpha}](\overline{x:\tau}).t) \\ \Delta, \overline{\alpha}; \Gamma, \overline{x:\tau} \vdash t : \tau' \rightsquigarrow \mathbf{t} \qquad \tau_{\mathbf{env}} = \langle (\Gamma(y_1))^{\mathcal{C}}, \ldots, (\Gamma(y_m))^{\mathcal{C}} \rangle \\ \mathbf{v} = \boldsymbol{\lambda}[\overline{\boldsymbol{\beta}}, \overline{\boldsymbol{\alpha}}](\mathbf{z} : \tau_{\mathbf{env}}, \overline{\mathbf{x} : \tau^{\mathcal{C}}}).(\mathbf{t}[\pi_1(\mathbf{z})/\mathbf{y_1}] \cdots [\pi_m(\mathbf{z})/\mathbf{y_m}]) \end{array}}{\Delta; \Gamma \vdash \lambda[\overline{\alpha}](\overline{x:\tau}).t : \forall[\overline{\alpha}].(\overline{\tau}) \to \tau' \rightsquigarrow \mathbf{pack}\langle \tau_{\mathbf{env}}, \langle \mathbf{v}[\overline{\beta}], \langle \overline{\mathbf{y}} \rangle \rangle \rangle \mathbf{as}\ \exists \alpha'.\langle (\forall[\overline{\alpha}].(\alpha', \overline{\tau}^{\mathcal{C}}) \to \tau'^{\mathcal{C}}), \alpha' \rangle}$$

Interoperability Semantics:

$$\mathbf{CF}^{\forall[\overline{\alpha}].(\overline{\tau}) \to \tau'}(\mathsf{v}) = \mathbf{pack}\langle \mathbf{unit}, \langle \mathbf{v}, () \rangle \rangle \mathbf{as}\ (\forall[\overline{\alpha}].(\overline{\tau}) \to \tau')^{\langle \mathcal{C} \rangle}$$

$$\text{where } \mathbf{v} = \boldsymbol{\lambda}[\overline{\alpha}](\mathbf{z} : \mathbf{unit}, \overline{\mathbf{x} : \tau^{\langle \mathcal{C} \rangle}[\alpha/\lceil \alpha \rceil]}).\mathcal{C}\mathcal{F}^{\tau'[\mathsf{L}\langle \alpha \rangle/\alpha]}(\mathsf{v}\ [\mathsf{L}\langle \alpha \rangle]\ \overline{\tau[\mathsf{L}\langle \alpha \rangle/\alpha]\mathcal{F}\mathcal{C}\mathbf{x}})$$

Perconti, J. T. & Ahmed, A.
Verifying an Open Compiler Using Multi-language Semantics *(ESOP)*, **2014**

Multi-Language Semantics in Compilation

Compiler:

$$\frac{\begin{array}{c} y_1, \ldots, y_m = \mathrm{fv}(\lambda[\overline{\alpha}](\overline{x:\tau}).t) \qquad \beta_1, \ldots, \beta_k = \mathrm{ftv}(\lambda[\overline{\alpha}](\overline{x:\tau}).t) \\ \Delta, \overline{\alpha}; \Gamma, \overline{x:\tau} \vdash t : \tau' \rightsquigarrow \mathbf{t} \qquad \tau_{\mathbf{env}} = \langle (\Gamma(y_1))^{\mathcal{C}}, \ldots, (\Gamma(y_m))^{\mathcal{C}} \rangle \\ \mathbf{v} = \boldsymbol{\lambda}[\overline{\boldsymbol{\beta}}, \overline{\boldsymbol{\alpha}}](\mathbf{z} : \tau_{\mathbf{env}}, \overline{\mathbf{x} : \tau^{\mathcal{C}}}).(\mathbf{t}[\pi_1(\mathbf{z})/\mathbf{y_1}] \cdots [\pi_m(\mathbf{z})/\mathbf{y_m}]) \end{array}}{\Delta; \Gamma \vdash \lambda[\overline{\alpha}](\overline{x:\tau}).t : \forall[\overline{\alpha}].(\overline{\tau}) \to \tau' \rightsquigarrow \mathbf{pack}\langle \tau_{\mathbf{env}}, \langle \mathbf{v}[\overline{\beta}], \langle \overline{\mathbf{y}} \rangle \rangle \rangle \mathbf{\ as\ } \exists \alpha'.\langle (\forall[\overline{\alpha}].(\alpha', \overline{\tau}^{\mathcal{C}}) \to \tau'^{\mathcal{C}}), \alpha' \rangle}$$

Interoperability Semantics:

$$\mathbf{CF}^{\forall[\overline{\alpha}].(\overline{\tau}) \to \tau'}(v) = \mathbf{pack}\langle \mathbf{unit}, \langle \mathbf{v}, () \rangle \rangle \mathbf{\ as\ } (\forall[\overline{\alpha}].(\overline{\tau}) \to \tau')^{\langle \mathcal{C} \rangle}$$

$$\text{where } \mathbf{v} = \boldsymbol{\lambda}[\overline{\boldsymbol{\alpha}}](\mathbf{z} : \mathbf{unit}, \overline{\mathbf{x} : \tau^{\langle \mathcal{C} \rangle}[\alpha/\lceil \alpha \rceil]}).\mathcal{C}\mathcal{F}^{\tau'[\mathsf{L}\langle\alpha\rangle/\alpha]}(v\ [\mathsf{L}\langle\alpha\rangle]\ \overline{\tau[\mathsf{L}\langle\alpha\rangle/\alpha]\mathcal{F}\mathcal{C}\mathbf{x}})$$

Perconti, J. T. & Ahmed, A.
Verifying an Open Compiler Using Multi-language Semantics *(ESOP)*, **2014**

Multi-Language Semantics in Compilation

Compiler:

$$\frac{\begin{array}{c} y_1, \ldots, y_m = \mathrm{fv}(\lambda[\overline{\alpha}](\overline{x:\tau}).t) \qquad \beta_1, \ldots, \beta_k = \mathrm{ftv}(\lambda[\overline{\alpha}](\overline{x:\tau}).t) \\ \Delta, \overline{\alpha}; \Gamma, \overline{x:\tau} \vdash t : \tau' \rightsquigarrow t \qquad \tau_{env} = \langle (\Gamma(y_1))^{\mathcal{C}}, \ldots, (\Gamma(y_m))^{\mathcal{C}} \rangle \\ v = \lambda[\overline{\beta}, \overline{\alpha}](z : \tau_{env}, \overline{x : \tau^{\mathcal{C}}}).(t[\pi_1(z)/y_1] \cdots [\pi_m(z)/y_m]) \end{array}}{\Delta; \Gamma \vdash \lambda[\overline{\alpha}](\overline{x:\tau}).t : \forall[\overline{\alpha}].(\overline{\tau}) \to \tau' \rightsquigarrow \mathbf{pack}\langle \tau_{env}, \langle v[\overline{\beta}], \langle \overline{y} \rangle \rangle \rangle \text{ as } \exists \alpha'.\langle (\forall[\overline{\alpha}].(\alpha', \overline{\tau}^{\mathcal{C}}) \to \tau'^{\mathcal{C}}), \alpha' \rangle}$$

Interoperability Semantics:

$$\mathbf{CF}^{\forall[\overline{\alpha}].(\overline{\tau}) \to \tau'}(v) = \mathbf{pack}\langle \mathbf{unit}, \langle v, () \rangle \rangle \text{ as } (\forall[\overline{\alpha}].(\overline{\tau}) \to \tau')^{\langle \mathcal{C} \rangle}$$

$$\text{where } v = \lambda[\overline{\alpha}](z : \mathbf{unit}, \overline{x : \tau^{\langle \mathcal{C} \rangle}[\alpha/\lceil \alpha \rceil]}).\mathcal{C}\mathcal{F}^{\tau'[\mathsf{L}\langle\alpha\rangle/\alpha]}(v\ [\mathsf{L}\langle\alpha\rangle]\ \overline{\tau[\mathsf{L}\langle\alpha\rangle/\alpha]\mathcal{F}\mathcal{C}x})$$

Perconti, J. T. & Ahmed, A.
Verifying an Open Compiler Using Multi-language Semantics *(ESOP)*, **2014**

Multi-Language Semantics in Compilation

Compiler:

$$\frac{\begin{array}{c} y_1, \ldots, y_m = \mathrm{fv}(\lambda[\overline{\alpha}](\overline{x:\tau}).t) \qquad \beta_1, \ldots, \beta_k = \mathrm{ftv}(\lambda[\overline{\alpha}](\overline{x:\tau}).t) \\ \Delta, \overline{\alpha}; \Gamma, \overline{x:\tau} \vdash t : \tau' \rightsquigarrow t \qquad \tau_{env} = \langle (\Gamma(y_1))^{\mathcal{C}}, \ldots, (\Gamma(y_m))^{\mathcal{C}} \rangle \\ v = \lambda[\overline{\beta}, \overline{\alpha}](z : \tau_{env}, \overline{x : \tau^{\mathcal{C}}}).(t[\pi_1(z)/y_1] \cdots [\pi_m(z)/y_m]) \end{array}}{\begin{array}{l} \Delta; \Gamma \vdash \lambda[\overline{\alpha}](\overline{x:\tau}).t : \forall[\overline{\alpha}].(\overline{\tau}) \to \tau' \rightsquigarrow \\ \qquad \mathbf{pack}\langle \tau_{env}, \langle v[\overline{\beta}], \langle \overline{y} \rangle \rangle \rangle \text{ as } \exists \alpha'.\langle (\forall[\overline{\alpha}].(\alpha', \overline{\tau}^{\mathcal{C}}) \to \tau'^{\mathcal{C}}), \alpha' \rangle \end{array}}$$

Interoperability Semantics:

$$\mathbf{CF}^{\forall[\overline{\alpha}].(\overline{\tau}) \to \tau'}(v) = \mathbf{pack}\langle \mathbf{unit}, \langle v, () \rangle \rangle \text{ as } (\forall[\overline{\alpha}].(\overline{\tau}) \to \tau')^{\langle \mathcal{C} \rangle}$$

$$\text{where } v = \lambda[\overline{\alpha}](z : \mathbf{unit}, \overline{x : \tau^{\langle \mathcal{C} \rangle}[\alpha/\lceil \alpha \rceil]}).\mathcal{C}\mathcal{F}^{\tau'[\overline{L\langle \alpha \rangle/\alpha}]}(v\ [\overline{L\langle \alpha \rangle}]\ \overline{\tau[L\langle \alpha \rangle/\alpha]\mathcal{F}\mathcal{C}x})$$

Perconti, J. T. & Ahmed, A.
Verifying an Open Compiler Using Multi-language Semantics *(ESOP)*, **2014**

Multi-Language Semantics in Compilation

Compiler:

$$\frac{\begin{array}{c} y_1, \ldots, y_m = \mathrm{fv}(\lambda[\overline{\alpha}](\overline{x:\tau}).t) \qquad \beta_1, \ldots, \beta_k = \mathrm{ftv}(\lambda[\overline{\alpha}](\overline{x:\tau}).t) \\ \Delta, \overline{\alpha}; \Gamma, \overline{x:\tau} \vdash t : \tau' \rightsquigarrow t \qquad \tau_{env} = \langle (\Gamma(y_1))^{\mathcal{C}}, \ldots, (\Gamma(y_m))^{\mathcal{C}} \rangle \\ v = \lambda[\overline{\beta}, \overline{\alpha}](z : \tau_{env}, \overline{x : \tau^{\mathcal{C}}}).(t[\pi_1(z)/y_1] \cdots [\pi_m(z)/y_m]) \end{array}}{\begin{array}{l} \Delta; \Gamma \vdash \lambda[\overline{\alpha}](\overline{x:\tau}).t : \forall[\overline{\alpha}].(\overline{\tau}) \to \tau' \rightsquigarrow \\ \qquad\qquad \mathbf{pack}\langle \tau_{env}, \langle v[\overline{\beta}], \langle \overline{y} \rangle \rangle \rangle \ \mathbf{as}\ \exists \alpha'.\langle (\forall[\overline{\alpha}].(\alpha', \overline{\tau}^{\mathcal{C}}) \to \tau'^{\mathcal{C}}), \alpha' \rangle \end{array}}$$

Interoperability Semantics:

$$\mathbf{CF}^{\forall[\overline{\alpha}].(\overline{\tau}) \to \tau'}(v) = \mathbf{pack}\langle \mathbf{unit}, \langle v, () \rangle \rangle \ \mathbf{as}\ (\forall[\overline{\alpha}].(\overline{\tau}) \to \tau')^{\langle \mathcal{C} \rangle}$$

$$\text{where } v = \lambda[\overline{\alpha}](z : \mathbf{unit}, \overline{x : \tau^{\langle \mathcal{C} \rangle}[\alpha/\lceil\alpha\rceil]}).\mathcal{C}\mathcal{F}^{\tau'}\overline{[L\langle\alpha\rangle/\alpha]}(v\ \overline{[L\langle\alpha\rangle]}\ \overline{\tau[L\langle\alpha\rangle/\alpha]}\mathcal{F}\mathcal{C}x)$$

Perconti, J. T. & Ahmed, A.
Verifying an Open Compiler Using Multi-language Semantics *(ESOP)*, **2014**

Multi-Language Semantics in Compilation

Compiler:

$$\frac{\begin{array}{c} y_1, \ldots, y_m = \mathrm{fv}(\lambda[\overline{\alpha}](\overline{x:\tau}).t) \qquad \beta_1, \ldots, \beta_k = \mathrm{ftv}(\lambda[\overline{\alpha}](\overline{x:\tau}).t) \\ \Delta, \overline{\alpha}; \Gamma, \overline{x:\tau} \vdash t : \tau' \rightsquigarrow \mathbf{t} \qquad \tau_{\mathbf{env}} = \langle (\Gamma(y_1))^{\mathcal{C}}, \ldots, (\Gamma(y_m))^{\mathcal{C}} \rangle \\ \mathbf{v} = \boldsymbol{\lambda}[\overline{\beta}, \overline{\alpha}](\mathbf{z} : \tau_{\mathbf{env}}, \overline{\mathbf{x} : \tau^{\mathcal{C}}}).(\mathbf{t}[\pi_1(\mathbf{z})/\mathbf{y_1}] \cdots [\pi_m(\mathbf{z})/\mathbf{y_m}]) \end{array}}{\begin{array}{l} \Delta; \Gamma \vdash \lambda[\overline{\alpha}](\overline{x:\tau}).t : \forall[\overline{\alpha}].(\overline{\tau}) \to \tau' \rightsquigarrow \\ \qquad \mathbf{pack}\langle \tau_{\mathbf{env}}, \langle \mathbf{v}[\overline{\beta}], \langle \overline{\mathbf{y}} \rangle \rangle \rangle \ \mathbf{as} \ \exists \alpha'.\langle (\forall[\overline{\alpha}].(\alpha', \overline{\tau}^{\mathcal{C}}) \to \tau'^{\mathcal{C}}), \alpha' \rangle \end{array}}$$

Interoperability Semantics:

$$\mathbf{CF}^{\forall[\overline{\alpha}].(\overline{\tau}) \to \tau'}(v) = \mathbf{pack}\langle \mathbf{unit}, \langle \mathbf{v}, () \rangle \rangle \ \mathbf{as} \ (\forall[\overline{\alpha}].(\overline{\tau}) \to \tau')^{\langle \mathcal{C} \rangle}$$

$$\text{where } \mathbf{v} = \boldsymbol{\lambda}[\overline{\alpha}](\mathbf{z} : \mathbf{unit}, \overline{\mathbf{x} : \tau^{\langle \mathcal{C} \rangle}[\alpha/\lceil\alpha\rceil]}).\mathcal{C}\mathcal{F}^{\tau'[\mathsf{L}\langle\alpha\rangle/\alpha]}(\mathbf{v}\ [\overline{\mathsf{L}\langle\alpha\rangle}]\ \overline{\tau[\mathsf{L}\langle\alpha\rangle/\alpha]\mathcal{F}\mathcal{C}\mathbf{x}})$$

Perconti, J. T. & Ahmed, A.
Verifying an Open Compiler Using Multi-language Semantics *(ESOP)*, **2014**

This is stupid

Why did we write the same thing twice?

Compiler:

$$\frac{\begin{array}{c}\mathsf{y}_1,\ldots,\mathsf{y}_m = \mathrm{fv}(\lambda[\overline{\alpha}](\overline{\mathsf{x}:\tau}).\mathsf{t}) \qquad \beta_1,\ldots,\beta_k = \mathrm{ftv}(\lambda[\overline{\alpha}](\overline{\mathsf{x}:\tau}).\mathsf{t}) \\ \Delta,\overline{\alpha};\Gamma,\overline{\mathsf{x}:\tau}\vdash \mathsf{t}:\tau' \rightsquigarrow \mathbf{t} \qquad \boldsymbol{\tau_{\mathrm{env}}} = \langle(\Gamma(\mathsf{y}_1))^{\mathcal{C}},\ldots,(\Gamma(\mathsf{y}_m))^{\mathcal{C}}\rangle \\ \mathbf{v} = \boldsymbol{\lambda}[\overline{\beta},\overline{\alpha}](\mathbf{z}:\boldsymbol{\tau_{\mathrm{env}}},\overline{\mathbf{x}:\tau^{\mathcal{C}}}).(\mathbf{t}[\boldsymbol{\pi}_1(\mathbf{z})/\mathsf{y}_1]\cdots[\boldsymbol{\pi}_m(\mathbf{z})/\mathsf{y}_m])\end{array}}{\Delta;\Gamma\vdash\lambda[\overline{\alpha}](\overline{\mathsf{x}:\tau}).\mathsf{t}:\forall[\overline{\alpha}].(\overline{\tau})\to\tau' \rightsquigarrow \mathbf{pack}\langle\boldsymbol{\tau_{\mathrm{env}}},\langle\mathbf{v}[\overline{\beta}],\langle\overline{\mathsf{y}}\rangle\rangle\rangle\ \mathbf{as}\ \exists\boldsymbol{\alpha}'.\langle(\forall[\overline{\alpha}].(\boldsymbol{\alpha}',\overline{\tau}^{\mathcal{C}})\to\tau'^{\mathcal{C}}),\boldsymbol{\alpha}'\rangle}$$

Worse, special case compiler:

$$\mathbf{CF}^{\forall[\overline{\alpha}].(\overline{\tau})\to\tau'}(\mathsf{v}) = \mathbf{pack}\langle\mathbf{unit},\langle\mathsf{v},()\rangle\rangle\ \mathbf{as}\ (\forall[\overline{\alpha}].(\overline{\tau})\to\tau')^{\langle\mathcal{C}\rangle}$$
$$\text{where } \mathbf{v} = \boldsymbol{\lambda}[\overline{\alpha}](\mathbf{z}:\mathbf{unit},\overline{\mathbf{x}:\tau^{\langle\mathcal{C}\rangle}[\alpha/\lceil\alpha\rceil]}).\mathcal{C}\mathcal{F}^{\tau'[\mathsf{L}\langle\alpha\rangle/\alpha]}(\mathsf{v}\ [\overline{\mathsf{L}\langle\alpha\rangle}]\ \overline{\tau[\mathsf{L}\langle\alpha\rangle/\alpha]\mathcal{F}\mathcal{C}\mathbf{x}})$$

Modelling Compilation as Multi-Language Semantics

Key Idea:

Start from multi-language semantic,
Derive the compiler!

```
p -> p'
```

$$\frac{p \to_s p}{p \to p'} \qquad \frac{p \to_t p}{p \to p'}$$

```
TS(false) -> 0
ST(0) -> false
```

Generalize source to target reductions from:
- closed values
+ to all, open terms…

Key Idea:

Start from multi-language semantic,
Derive the compiler!

```
p -> p'
```

$$\frac{p \rightarrow_s p}{p \rightarrow p'} \qquad \frac{p \rightarrow_t p}{p \rightarrow p'}$$

```
TS(false) -> 0
ST(0) -> false
```

Generalize source to target reductions from:
- closed values
+ to all, open terms…
+ pick a reduction strategy
= compiler

Key Idea:

Start from multi-language semantic,
Derive the compiler***S***!

```
p -> p'
```

$$\frac{p \to_s p}{p \to p'} \qquad \frac{p \to_t p}{p \to p'}$$

```
TS(false) -> 0
ST(0) -> false
```

Full normalization of all TS redexes
= AOT compilation.

CBV in the multi-language
= JIT compilation (ish).

ANF Translation as Multi-Language Semantics

Thesis: If we start from an interoperability semantics, we can derive a correct compiler.

ANF Translation as Multi-Language Semantics

Thesis: If we start from an interoperability semantics, we can derive a correct compiler.

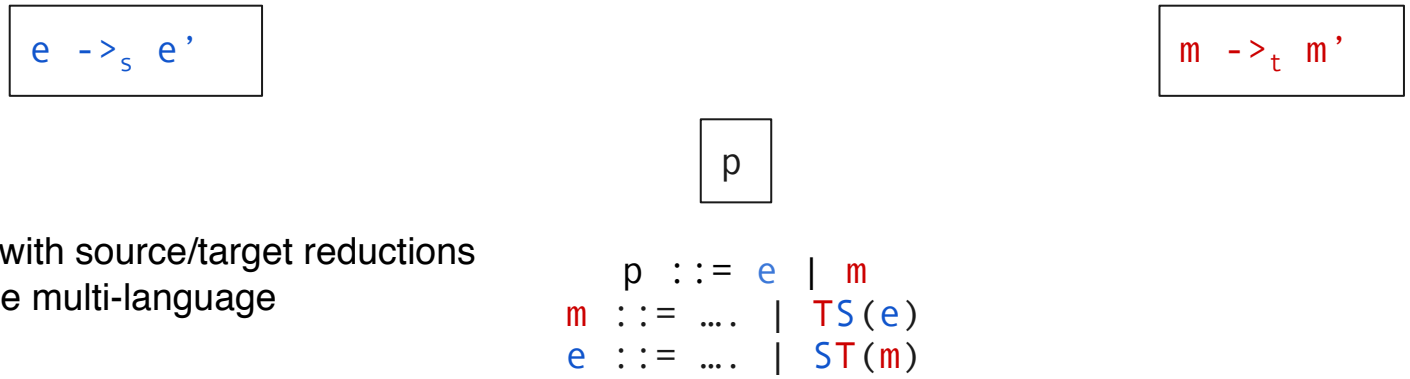
$e \to_s e'$

$m \to_t m'$

1. Start with source/target reductions

ANF Translation as Multi-Language Semantics

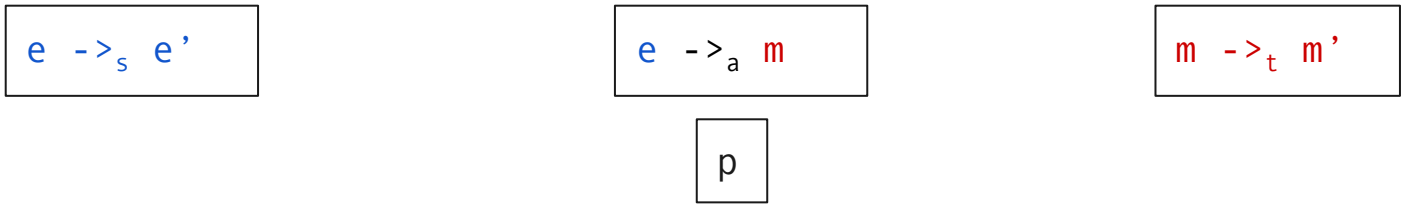
Thesis: If we start from an interoperability semantics, we can derive a correct compiler.



1. Start with source/target reductions
2. Create multi-language

ANF Translation as Multi-Language Semantics

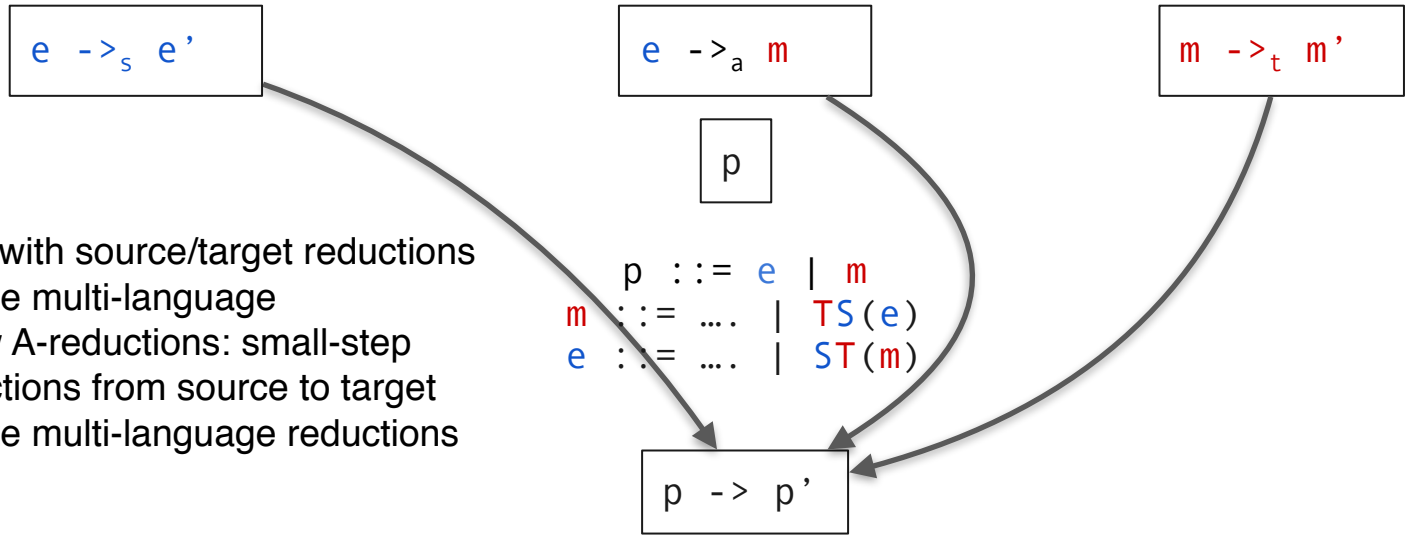
Thesis: If we start from an interoperability semantics, we can derive a correct compiler.



1. Start with source/target reductions
2. Create multi-language
3. Show A-reductions: small-step reductions from source to target

ANF Translation as Multi-Language Semantics

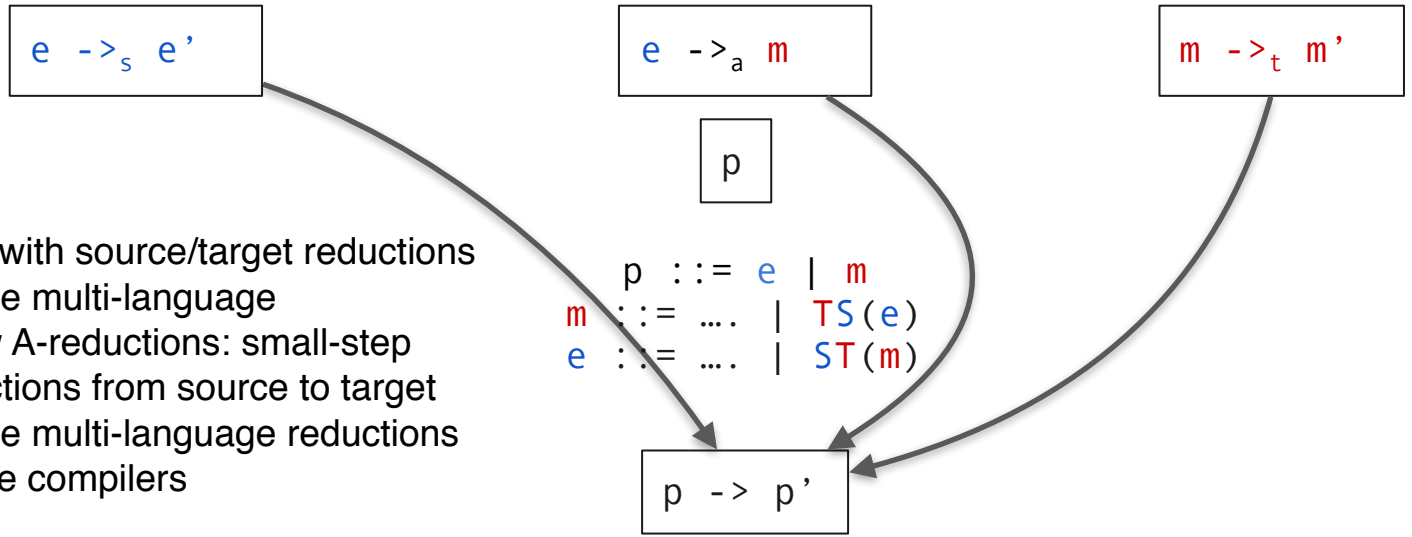
Thesis: If we start from an interoperability semantics, we can derive a correct compiler.



1. Start with source/target reductions
2. Create multi-language
3. Show A-reductions: small-step reductions from source to target
4. Create multi-language reductions

ANF Translation as Multi-Language Semantics

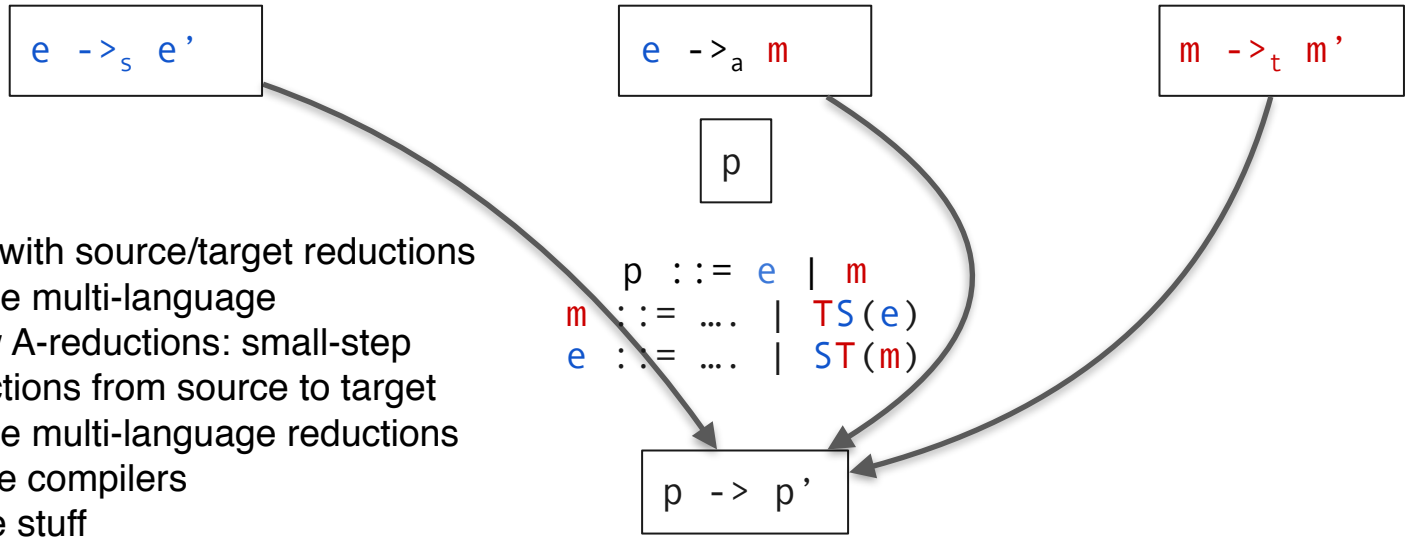
Thesis: If we start from an interoperability semantics, we can derive a correct compiler.



1. Start with source/target reductions
2. Create multi-language
3. Show A-reductions: small-step reductions from source to target
4. Create multi-language reductions
5. Derive compilers

ANF Translation as Multi-Language Semantics

Thesis: If we start from an interoperability semantics, we can derive a correct compiler.



1. Start with source/target reductions
2. Create multi-language
3. Show A-reductions: small-step reductions from source to target
4. Create multi-language reductions
5. Derive compilers
6. Prove stuff

Source:

$$
\begin{array}{rcl}
e & ::= & x \mid (\lambda x.e) \mid (e\ e) \mid (\text{set!}\ x\ e) \mid (\text{let}\ ([x\ e]\ ...)\ e) \mid (\text{begin}\ e\ ...)\ \mid .... \\
 & & \mid SA(M)
\end{array}
$$

Target:

$$
\begin{array}{rcl}
V & ::= & x \mid (\lambda x.M) \mid \text{true} \mid \text{false} \mid .... \\
 & & AS(e) \\
N & ::= & V \mid (V\ V) \mid (\text{set!}\ x\ V) \mid .... \\
 & & AS(e) \\
M & ::= & N \mid (\text{let}\ ([x\ N]\ ...)\ M) \mid (\text{begin}\ N\ ...\ M) \mid (\text{if}\ V\ M\ M) \mid .... \\
 & & AS(e)
\end{array}
$$

Source:

$$
\begin{array}{rcl}
e & ::= & x \mid (\lambda x.e) \mid (e\ e) \mid (\text{set!}\ x\ e) \mid (\text{let}\ ([x\ e]\ ...)\ e) \mid (\text{begin}\ e\ ...)\ \mid .... \\
 & & \mid SA(M)
\end{array}
$$

Target:

$$
\begin{array}{rcl}
V & ::= & x \mid (\lambda x.M) \mid \text{true} \mid \text{false} \mid .... \\
 & & AS(e) \\
N & ::= & V \mid (V\ V) \mid (\text{set!}\ x\ V) \mid .... \\
 & & AS(e) \\
M & ::= & N \mid (\text{let}\ ([x\ N]\ ...)\ M) \mid (\text{begin}\ N\ ...\ M) \mid (\text{if}\ V\ M\ M) \mid .... \\
 & & AS(e)
\end{array}
$$

Source:

$$
\begin{array}{rcl}
e & ::= & x \mid (\lambda x.e) \mid (e\ e) \mid (\text{set!}\ x\ e) \mid (\text{let}\ ([x\ e]\ ...)\ e) \mid (\text{begin}\ e\ ...)\ \mid .... \\
 & & \mid SA(M)
\end{array}
$$

Target:

$$
\begin{array}{rcl}
V & ::= & x \mid (\lambda x.M) \mid \text{true} \mid \text{false} \mid .... \\
 & & AS(e) \\
N & ::= & V \mid (V\ V) \mid (\text{set!}\ x\ V) \mid .... \\
 & & AS(e) \\
M & ::= & N \mid (\text{let}\ ([x\ N]\ ...)\ M) \mid (\text{begin}\ N\ ...\ M) \mid (\text{if}\ V\ M\ M) \mid .... \\
 & & AS(e)
\end{array}
$$

Source:

$$
\begin{array}{rcl}
e & ::= & x \mid (\lambda x.e) \mid (e\ e) \mid (\text{set!}\ x\ e) \mid (\text{let}\ ([x\ e]\ ...)\ e) \mid (\text{begin}\ e\ ...)\ \mid .... \\
 & & \mid SA(M)
\end{array}
$$

Target:

$$
\begin{array}{rcl}
V & ::= & x \mid (\lambda x.M) \mid \text{true} \mid \text{false} \mid .... \\
 & & AS(e) \\
N & ::= & V \mid (V\ V) \mid (\text{set!}\ x\ V) \mid .... \\
 & & AS(e) \\
M & ::= & N \mid (\text{let}\ ([x\ N]\ ...)\ M) \mid (\text{begin}\ N\ ...\ M) \mid (\text{if}\ V\ M\ M) \mid .... \\
 & & AS(e)
\end{array}
$$

Source:

$$
\begin{array}{rcl}
e & ::= & x \mid (\lambda x.e) \mid (e\ e) \mid (\text{set!}\ x\ e) \mid (\text{let}\ ([x\ e]\ ...)\ e) \mid (\text{begin}\ e\ ...)\ \mid .... \\
 & & \mid SA(M)
\end{array}
$$

Target:

$$
\begin{array}{rcl}
V & ::= & x \mid (\lambda x.M) \mid \text{true} \mid \text{false} \mid .... \\
 & & AS(e) \\
N & ::= & V \mid (V\ V) \mid (\text{set!}\ x\ V) \mid .... \\
 & & AS(e) \\
M & ::= & N \mid (\text{let}\ ([x\ N]\ ...)\ M) \mid (\text{begin}\ N\ ...\ M) \mid (\text{if}\ V\ M\ M) \mid .... \\
 & & AS(e)
\end{array}
$$

Source:

$$
\begin{array}{rcl}
e & ::= & x \mid (\lambda x.e) \mid (e\ e) \mid (\text{set!}\ x\ e) \mid (\text{let}\ ([x\ e]\ ...)\ e) \mid (\text{begin}\ e\ ...)\ \mid .... \\
 & & \mid SA(M)
\end{array}
$$

Target:

$$
\begin{array}{rcl}
V & ::= & x \mid (\lambda x.M) \mid \text{true} \mid \text{false} \mid .... \\
 & & AS(e) \\
N & ::= & V \mid (V\ V) \mid (\text{set!}\ x\ V) \mid .... \\
 & & AS(e) \\
M & ::= & N \mid (\text{let}\ ([x\ N]\ ...)\ M) \mid (\text{begin}\ N\ ...\ M) \mid (\text{if}\ V\ M\ M) \mid .... \\
 & & AS(e)
\end{array}
$$

Source:

$$
\begin{array}{rcl}
e & ::= & x \mid (\lambda x.e) \mid (e\ e) \mid (\text{set!}\ x\ e) \mid (\text{let}\ ([x\ e]\ ...)\ e) \mid (\text{begin}\ e\ ...)\ \mid .... \\
 & & \mid SA(M)
\end{array}
$$

Note: $M \subset e$
Even without boundaries

Target:

$$
\begin{array}{rcl}
V & ::= & x \mid (\lambda x.M) \mid \text{true} \mid \text{false} \mid .... \\
 & & AS(e) \\
N & ::= & V \mid (V\ V) \mid (\text{set!}\ x\ V) \mid .... \\
 & & AS(e) \\
M & ::= & N \mid (\text{let}\ ([x\ N]\ ...)\ M) \mid (\text{begin}\ N\ ...\ M) \mid (\text{if}\ V\ M\ M) \mid .... \\
 & & AS(e)
\end{array}
$$

Source:

$$
\begin{array}{rcl}
v & ::= & (\lambda x.e) \mid true \mid .... \\
E & ::= & [\cdot] \mid E\ e \mid v\ E \mid .... \\
H & ::= & \cdot \mid H, x \mapsto v
\end{array}
$$

$$\boxed{H;\ e \to H;\ e}$$

$$
\begin{array}{rcl}
(\lambda x.e)\ v & \to & e[x\ := v] \\
& \vdots &
\end{array}
\qquad
\frac{e \to e'}{E[e] \to E[e']}
$$

Target:

$$E \quad ::= \quad [\cdot] \mid (\text{let}([x\ V]\ ...[x\ [\cdot]][x\ N]\ ...)\ M)$$

$$\boxed{H;\ M \to H;\ M}$$

$$
\begin{array}{rcl}
(\lambda x.M)\ V & \to & M[x\ := V] \\
& \vdots & \\
(\text{let}\ ([x\ V]\ ...)\ M) & \to & M[x := V\ ...]
\end{array}
$$

Source:

$$\begin{array}{rcl} v & ::= & (\lambda x.e) \mid true \mid .... \\ E & ::= & [\cdot] \mid E\ e \mid v\ E \mid .... \\ H & ::= & \cdot \mid H, x \mapsto v \end{array}$$

$$\boxed{H;\ e \to H;\ e}$$

$$\begin{array}{rcl} (\lambda x.e)\ v & \to & e[x\ := v] \\ & \vdots & \end{array}$$

$$\frac{e \to e'}{E[e] \to E[e']}$$

Target:

$$E \quad ::= \quad [\cdot] \mid (\text{let}([x\ V]\ ...[x\ [\cdot]][x\ N]\ ...)\ M)$$

$$\boxed{H;\ M \to H;\ M}$$

$$\begin{array}{rcl} (\lambda x.M)\ V & \to & M[x\ := V] \\ & \vdots & \\ (\text{let}\ ([x\ V]\ ...)\ M) & \to & M[x := V\ ...] \end{array}$$

Source:

$$\begin{array}{rcl} v & ::= & (\lambda x.e) \mid true \mid .... \\ E & ::= & [\cdot] \mid E\ e \mid v\ E \mid .... \\ H & ::= & \cdot \mid H, x \mapsto v \end{array}$$

$$\boxed{H;\ e \to H;\ e}$$

$$\begin{array}{rcl} (\lambda x.e)\ v & \to & e[x\ := v] \\ & \vdots & \end{array} \qquad \frac{e \to e'}{E[e] \to E[e']}$$

Target:

$$E \quad ::= \quad [\cdot] \mid (\text{let}([x\ V] ...[x\ [\cdot]][x\ N] ...)\ M)$$

$$\boxed{H;\ M \to H;\ M}$$

$$\begin{array}{rcl} (\lambda x.M)\ V & \to & M[x\ := V] \\ & \vdots & \\ (\text{let}\ ([x\ V] ...)\ M) & \to & M[x := V\ ...] \end{array}$$

Source:

$$\begin{array}{rcl} v & ::= & (\lambda x.e) \mid true \mid .... \\ E & ::= & [\cdot] \mid E\ e \mid v\ E \mid .... \\ H & ::= & \cdot \mid H, x \mapsto v \end{array}$$

$\boxed{H;\ e \to H;\ e}$

$$\begin{array}{rcl} (\lambda x.e)\ v & \to & e[x\ := v] \\ & \vdots & \end{array} \qquad \frac{e \to e'}{E[e] \to E[e']}$$

Target:

$$E \quad ::= \quad [\cdot] \mid (\text{let}([x\ V] ...[x\ [\cdot]][x\ N] ...)\ M)$$

$\boxed{H;\ M \to H;\ M}$

$$\begin{array}{rcl} (\lambda x.M)\ V & \to & M[x\ := V] \\ & \vdots & \\ (\text{let}\ ([x\ V] ...)\ M) & \to & M[x := V\ ...] \end{array}$$

A reductions

adapted from [1]

$e \to^a e$

$$
\begin{array}{rcll}
M & \to^a & SA(M) & \text{[A-normal]} \\
E[(\text{let } ([x\ e] \ldots)\ e_2] & \to^a & SA(\text{let } ([x\ AS(e)] \ldots)\ AS(E[e_2])) & \text{[A-merge-let]} \\
E[(\text{begin } e_1 \ldots e_2] & \to^a & SA(\text{begin } AS(e_1) \ldots AS(E[e_2])) & \text{[A-merge-begin]} \\
E[N] & \to^a & SA(\text{let } ([x\ N])\ AS(E[x])) & \text{[A-lift]} \\
 & & \text{where } E \text{ is not an ``}N\text{ accepting target context''} & \\
 & & \text{and } N \text{ is not a } V & \\
 & \vdots & &
\end{array}
$$

[1] Flanagan, C.; Sabry, A.; Duba, B. F. & Felleisen, M. The Essence of Compiling with Continuations *(PLDI)*, **1993.**

A reductions

$e \to^a e$

$$
\begin{array}{rcll}
M & \to^a & SA(M) & \text{[A-normal]} \\
E[(\text{let } ([x\ e] \ldots)\ e_2] & \to^a & SA(\text{let } ([x\ AS(e)] \ldots)\ AS(E[e_2])) & \text{[A-merge-let]} \\
E[(\text{begin } e_1 \ldots e_2] & \to^a & SA(\text{begin } AS(e_1) \ldots AS(E[e_2])) & \text{[A-merge-begin]} \\
E[N] & \to^a & SA(\text{let } ([x\ N])\ AS(E[x])) & \text{[A-lift]} \\
 & & \text{where } E \text{ is not an ``}N\text{ accepting target context''} & \\
 & & \text{and } N \text{ is not a } V & \\
 & \vdots & &
\end{array}
$$

[1] Flanagan, C.; Sabry, A.; Duba, B. F. & Felleisen, M. The Essence of Compiling with Continuations *(PLDI)*, **1993.**

A reductions

$\boxed{e \to^a e}$

$$
\begin{array}{rcll}
M & \to^a & SA(M) & \text{[A-normal]} \\
E[(\text{let } ([x\ e] \ldots)\ e_2] & \to^a & SA(\text{let } ([x\ AS(e)] \ldots)\ AS(E[e_2])) & \text{[A-merge-let]} \\
E[(\text{begin } e_1 \ldots e_2] & \to^a & SA(\text{begin } AS(e_1) \ldots AS(E[e_2])) & \text{[A-merge-begin]} \\
E[N] & \to^a & SA(\text{let } ([x\ N])\ AS(E[x])) & \text{[A-lift]} \\
 & & \text{where } E \text{ is not an ``}N\text{ accepting target context''} & \\
 & & \text{and } N \text{ is not a } V & \\
 & \vdots & &
\end{array}
$$

[1] Flanagan, C.; Sabry, A.; Duba, B. F. & Felleisen, M. The Essence of Compiling with Continuations *(PLDI)*, **1993.**

A reductions

```
(e >>= e2(x)) >>= E([.])  ⟹  e >>= (e2(x) >>= E[.])
```

$e \rightarrow^a e$

$$
\begin{array}{rcll}
M & \rightarrow^a & SA(M) & \text{[A-normal]} \\
E[(\text{let } ([x\ e] \ldots)\ e_2] & \rightarrow^a & SA(\text{let } ([x\ AS(e)] \ldots)\ AS(E[e_2])) & \text{[A-merge-let]} \\
E[(\text{begin } e_1 \ldots e_2] & \rightarrow^a & SA(\text{begin } AS(e_1) \ldots AS(E[e_2])) & \text{[A-merge-begin]} \\
E[N] & \rightarrow^a & SA(\text{let } ([x\ N])\ AS(E[x])) & \text{[A-lift]} \\
 & & \text{where } E \text{ is not an ``}N\text{ accepting target context''} & \\
 & & \text{and } N \text{ is not a } V & \\
 & \vdots & &
\end{array}
$$

[1] Flanagan, C.; Sabry, A.; Duba, B. F. & Felleisen, M. The Essence of Compiling with Continuations *(PLDI)*, **1993.**

A reductions

```
(e >>= e2(x)) >>= E([.])  ⟹  e >>= (e2(x) >>= E[.])
```

$e \rightarrow^a e$

$$
\begin{array}{rcll}
M & \rightarrow^a & SA(M) & \text{[A-normal]} \\
E[(\text{let } ([x\ e] \ldots)\ e_2] & \rightarrow^a & SA(\text{let } ([x\ AS(e)] \ldots)\ AS(E[e_2])) & \text{[A-merge-let]} \\
E[(\text{begin } e_1 \ldots e_2] & \rightarrow^a & SA(\text{begin } AS(e_1) \ldots AS(E[e_2])) & \text{[A-merge-begin]} \\
E[N] & \rightarrow^a & SA(\text{let } ([x\ N])\ AS(E[x])) & \text{[A-lift]} \\
 & & \text{where } E \text{ is not an ``}N\text{ accepting target context''} & \\
 & & \text{and } N \text{ is not a } V & \\
 & & \vdots &
\end{array}
$$

[1] Flanagan, C.; Sabry, A.; Duba, B. F. & Felleisen, M. The Essence of Compiling with Continuations *(PLDI)*, **1993.**

A reductions

$\boxed{e \to^a e}$

$$
\begin{array}{rcll}
M & \to^a & SA(M) & \text{[A-normal]} \\
E[(\text{let } ([x\ e] \ldots)\ e_2] & \to^a & SA(\text{let } ([x\ AS(e)] \ldots)\ AS(E[e_2])) & \text{[A-merge-let]} \\
E[(\text{begin } e_1 \ldots e_2] & \to^a & SA(\text{begin } AS(e_1) \ldots AS(E[e_2])) & \text{[A-merge-begin]} \\
E[N] & \to^a & SA(\text{let } ([x\ N])\ AS(E[x])) & \text{[A-lift]} \\
 & & \text{where } E \text{ is not an ``}N\text{ accepting target context''} & \\
 & & \text{and } N \text{ is not a } V & \\
 & \vdots & &
\end{array}
$$

[1] Flanagan, C.; Sabry, A.; Duba, B. F. & Felleisen, M. The Essence of Compiling with Continuations *(PLDI)*, **1993.**

A reductions

$e \to^a e$

$$
\begin{array}{rcll}
M & \to^a & SA(M) & \text{[A-normal]} \\
E[(\text{let } ([x\ e] \ldots)\ e_2] & \to^a & SA(\text{let } ([x\ AS(e)] \ldots)\ AS(E[e_2])) & \text{[A-merge-let]} \\
E[(\text{begin } e_1 \ldots e_2] & \to^a & SA(\text{begin } AS(e_1) \ldots AS(E[e_2])) & \text{[A-merge-begin]} \\
E[N] & \to^a & SA(\text{let } ([x\ N])\ AS(E[x])) & \text{[A-lift]} \\
 & & \text{where } E \text{ is not an ``}N\text{ accepting target context''} & \\
 & & \text{and } N \text{ is not a } V & \\
 & & \vdots &
\end{array}
$$

[1] Flanagan, C.; Sabry, A.; Duba, B. F. & Felleisen, M. The Essence of Compiling with Continuations *(PLDI)*, **1993.**

A reductions

Example N accepting context:

(let ([x [$\cdot$]) M)

$e \rightarrow^a e$

$$
\begin{array}{rcll}
M & \rightarrow^a & SA(M) & \text{[A-normal]} \\
E[(\text{let } ([x\ e] \ldots)\ e_2] & \rightarrow^a & SA(\text{let } ([x\ AS(e)] \ldots)\ AS(E[e_2])) & \text{[A-merge-let]} \\
E[(\text{begin } e_1 \ldots\ e_2] & \rightarrow^a & SA(\text{begin } AS(e_1) \ldots\ AS(E[e_2])) & \text{[A-merge-begin]} \\
E[N] & \rightarrow^a & SA(\text{let } ([x\ N])\ AS(E[x])) & \text{[A-lift]} \\
 & & \text{where } E \text{ is not an ``}N\text{ accepting target context''} & \\
 & & \text{and } N \text{ is not a } V &
\end{array}
$$

[1] Flanagan, C.; Sabry, A.; Duba, B. F. & Felleisen, M. The Essence of Compiling with Continuations *(PLDI)*, **1993.**

A reductions

$\boxed{e \to^a e}$

$$
\begin{array}{rcll}
M & \to^a & SA(M) & \text{[A-normal]} \\
E[(\text{let } ([x\ e] \ldots)\ e_2] & \to^a & SA(\text{let } ([x\ AS(e)] \ldots)\ AS(E[e_2])) & \text{[A-merge-let]} \\
E[(\text{begin } e_1 \ldots e_2] & \to^a & SA(\text{begin } AS(e_1) \ldots AS(E[e_2])) & \text{[A-merge-begin]} \\
E[N] & \to^a & SA(\text{let } ([x\ N])\ AS(E[x])) & \text{[A-lift]} \\
 & & \text{where } E \text{ is not an ``}N\text{ accepting target context''} & \\
 & & \text{and } N \text{ is not a } V & \\
 & \vdots & &
\end{array}
$$

[1] Flanagan, C.; Sabry, A.; Duba, B. F. & Felleisen, M. The Essence of Compiling with Continuations *(PLDI)*, **1993.**

DEMO TIME

Single-Step Translate SOME Sub-term

$$
\begin{array}{rcl}
p & ::= & e \mid M \\
T & ::= & AS(C_M) \\
C_M & ::= & [\cdot] \mid (\text{let } ([x\ N] \ldots) C_M) \mid \ldots.
\end{array}
$$

$$\boxed{M \xrightarrow{s}{}^{a} M}$$

$$\frac{p \rightarrow^{a} p'}{C[T[p]] \xrightarrow{s}{}^{a} C[T[p']]} \qquad \frac{}{C[SA(AS(e))] \xrightarrow{s}{}^{a} C[e]}$$

$$\frac{}{C[AS(SA(M))] \xrightarrow{s}{}^{a} C[M]}$$

Single-Step Translate SOME Sub-term

$$
\begin{array}{rcl}
p & ::= & e \mid M \\
T & ::= & AS(C_M) \\
C_M & ::= & [\cdot] \mid (\text{let } ([x\ N] \ldots) C_M) \mid \ldots.
\end{array}
$$

$$\boxed{M \xrightarrow{s}{}^{a} M}$$

$$\frac{p \rightarrow^{a} p'}{C[T[p]] \xrightarrow{s}{}^{a} C[T[p']]} \qquad \frac{}{C[SA(AS(e))] \xrightarrow{s}{}^{a} C[e]}$$

$$\frac{}{C[AS(SA(M))] \xrightarrow{s}{}^{a} C[M]}$$

Single-Step Translate SOME Sub-term

$$
\begin{array}{rcl}
p & ::= & e \mid M \\
T & ::= & AS(C_M) \\
C_M & ::= & [\cdot] \mid (\text{let } ([x\ N] \ldots) C_M) \mid \ldots.
\end{array}
$$

$$\boxed{M \xrightarrow{s}{}^{a} M}$$

$$\frac{p \rightarrow^{a} p'}{C[T[p]] \xrightarrow{s}{}^{a} C[T[p']]} \qquad \frac{}{C[SA(AS(e))] \xrightarrow{s}{}^{a} C[e]}$$

$$\frac{}{C[AS(SA(M))] \xrightarrow{s}{}^{a} C[M]}$$

Single-Step Translate SOME Sub-term

$$
\begin{array}{rcl}
p & ::= & e \mid M \\
T & ::= & AS(C_M) \\
C_M & ::= & [\cdot] \mid (\text{let } ([x\ N] \ldots) C_M) \mid \ldots
\end{array}
$$

$$\boxed{M \xrightarrow{s}{}^{a} M}$$

$$\frac{p \rightarrow^{a} p'}{C[T[p]] \xrightarrow{s}{}^{a} C[T[p']]} \qquad \frac{}{C[SA(AS(e))] \xrightarrow{s}{}^{a} C[e]}$$

$$\frac{}{C[AS(SA(M))] \xrightarrow{s}{}^{a} C[M]}$$

Single-Step Translate SOME Sub-term

$$
\begin{array}{rcl}
p & ::= & e \mid M \\
T & ::= & AS(C_M) \\
C_M & ::= & [\cdot] \mid (\text{let } ([x\ N] \ldots) C_M) \mid \ldots.
\end{array}
$$

$$\boxed{M \xrightarrow{s}{}^{a} M}$$

$$\frac{p \rightarrow^{a} p'}{C[T[p]] \xrightarrow{s}{}^{a} C[T[p']]} \qquad \frac{}{C[SA(AS(e))] \xrightarrow{s}{}^{a} C[e]}$$

$$\frac{}{C[AS(SA(M))] \xrightarrow{s}{}^{a} C[M]}$$

The Multi-Language JIT

$$\boxed{H;\ p \to^{p} H;\ p}$$

$$\frac{H\ e \to H';\ e'}{H;\ e \to^{p} H';\ e'}\ \textsc{Step-S1} \qquad \frac{H\ e \to H';\ e'}{H;\ AS(e) \to^{p} H';\ AS(e')}\ \textsc{Step-S2}$$

$$\frac{H\ M \to H';\ M'}{H;\ M \to^{p} H';\ M'}\ \textsc{Step-T1} \qquad \frac{H\ M \to H';\ M'}{H;\ SA(M) \to^{p} H';\ SA(M')}\ \textsc{Step-T2}$$

$$\frac{M\ {}^{s}\!\to^{a} M'}{H;\ M \to^{p} H;\ M'}\ \textsc{Step-Across}$$

The Multi-Language JIT

$$\boxed{H;\ p \to^{p} H;\ p}$$

$$\frac{H\ e \to H';\ e'}{H;\ e \to^{p} H';\ e'}\ \textsc{Step-S1} \qquad \frac{H\ e \to H';\ e'}{H;\ AS(e) \to^{p} H';\ AS(e')}\ \textsc{Step-S2}$$

$$\frac{H\ M \to H';\ M'}{H;\ M \to^{p} H';\ M'}\ \textsc{Step-T1} \qquad \frac{H\ M \to H';\ M'}{H;\ SA(M) \to^{p} H';\ SA(M')}\ \textsc{Step-T2}$$

$$\frac{M\ {}^{s}\!\to^{a} M'}{H;\ M \to^{p} H;\ M'}\ \textsc{Step-Across}$$

The Multi-Language JIT

$$\boxed{H;\ p \to^{p} H;\ p}$$

$$\frac{H\ e \to H';\ e'}{H;\ e \to^{p} H';\ e'}\ \textsc{Step-S1} \qquad \frac{H\ e \to H';\ e'}{H;\ AS(e) \to^{p} H';\ AS(e')}\ \textsc{Step-S2}$$

$$\frac{H\ M \to H';\ M'}{H;\ M \to^{p} H';\ M'}\ \textsc{Step-T1} \qquad \frac{H\ M \to H';\ M'}{H;\ SA(M) \to^{p} H';\ SA(M')}\ \textsc{Step-T2}$$

$$\frac{M\ {}^{s}\!\to^{a} M'}{H;\ M \to^{p} H;\ M'}\ \textsc{Step-Across}$$

The Multi-Language JIT

$$\boxed{H;\ p \to^{p} H;\ p}$$

$$\frac{H\ e \to H';\ e'}{H;\ e \to^{p} H';\ e'}\ \textsc{Step-S1} \qquad \frac{H\ e \to H';\ e'}{H;\ AS(e) \to^{p} H';\ AS(e')}\ \textsc{Step-S2}$$

$$\frac{H\ M \to H';\ M'}{H;\ M \to^{p} H';\ M'}\ \textsc{Step-T1} \qquad \frac{H\ M \to H';\ M'}{H;\ SA(M) \to^{p} H';\ SA(M')}\ \textsc{Step-T2}$$

$$\frac{M\ {}^{s}\!\to^{a} M'}{H;\ M \to^{p} H;\ M'}\ \textsc{Step-Across}$$

The Multi-Language JIT

$$\boxed{H;\ p \to^{p} H;\ p}$$

$$\frac{H\ e \to H';\ e'}{H;\ e \to^{p} H';\ e'}\ \textsc{Step-S1} \qquad \frac{H\ e \to H';\ e'}{H;\ AS(e) \to^{p} H';\ AS(e')}\ \textsc{Step-S2}$$

$$\frac{H\ M \to H';\ M'}{H;\ M \to^{p} H';\ M'}\ \textsc{Step-T1} \qquad \frac{H\ M \to H';\ M'}{H;\ SA(M) \to^{p} H';\ SA(M')}\ \textsc{Step-T2}$$

$$\frac{M\ {}^{s}\!\to^{a} M'}{H;\ M \to^{p} H;\ M'}\ \textsc{Step-Across}$$

JIT DEMO TIME

Derive the AOT Compiler

$$\boxed{e \Downarrow_{AOT} M}$$

$$\frac{AS(e) \overset{s}{\rightarrow}{}^{a}{}_{*}\ M \qquad M \overset{s}{\not\rightarrow}{}^{a}\ M'}{e \Downarrow_{AOT} M}$$

Derive the AOT Compiler

$$\boxed{e \Downarrow_{AOT} M}$$

$$\frac{AS(e) \overset{s}{\rightarrow}{}^{a}{}_{*}\ M \qquad M \overset{s}{\not\rightarrow}{}^{a}\ M'}{e \Downarrow_{AOT} M}$$

Derive the AOT Compiler

$$\boxed{e \Downarrow_{AOT} M}$$

$$\frac{AS(e) \ {}^{s}\!\rightarrow^{a}\!* \ M \qquad M \ {}^{s}\!\not\rightarrow^{a} \ M'}{e \Downarrow_{AOT} M}$$

META THEORY

From Confluence, Correctness

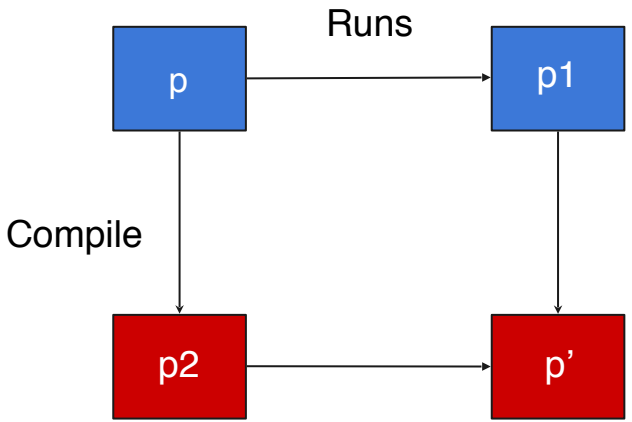
- Correctness follows confluence

Proposition 1 (Confluence)
If $H; p \rightarrow^{p*} H_1; p_1$ *and* $H; p \rightarrow^{p*} H_2; p_2$ *then* $H_1; p_1 \rightarrow^{p*} H'; p'$ *and* $H_2; p_2 \rightarrow^{p*} H'; p'$.

From Confluence, Correctness

- Correctness follows confluence
- Intuitively, confluence is the same diagram as correctness

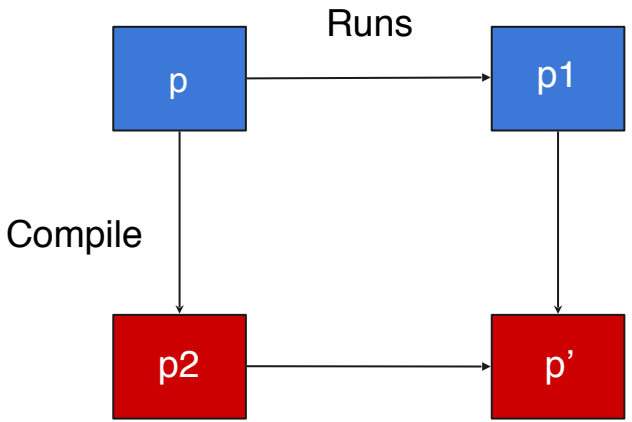
Proposition 1 (Confluence)
If $H; p \rightarrow^{p*} H_1; p_1$ *and* $H; p \rightarrow^{p*} H_2; p_2$ *then* $H_1; p_1 \rightarrow^{p*} H'; p'$ *and* $H_2; p_2 \rightarrow^{p*} H'; p'$.



From Confluence, Correctness

- Correctness follows confluence
- Intuitively, confluence is the same diagram as correctness
- Multi-language reduction includes interpreter and compiler, nondeterministic choice.

Proposition 1 (Confluence)
If $H; p \rightarrow^{p*} H_1; p_1$ *and* $H; p \rightarrow^{p*} H_2; p_2$ *then* $H_1; p_1 \rightarrow^{p*} H'; p'$ *and* $H_2; p_2 \rightarrow^{p*} H'; p'$.



From Confluence, Correctness

- Correctness follows confluence
- Intuitively, confluence is the same diagram as correctness
- Multi-language reduction includes interpreter and compiler, nondeterministic choice.

$$\boxed{e \Downarrow_{AOT} M}$$

$$\frac{AS(e) \ {}^{s}\!\rightarrow^{a}\!* \ M \qquad M \ {}^{s}\!\not\rightarrow^{a} \ M'}{e \Downarrow_{AOT} M}$$

Proposition 1 (Confluence)
If $H; p \rightarrow^{p*} H_1; p_1$ *and* $H; p \rightarrow^{p*} H_2; p_2$ *then* $H_1; p_1 \rightarrow^{p*} H'; p'$ *and* $H_2; p_2 \rightarrow^{p*} H'; p'$.

Corollary 1 (Whole-Program Correctness)
If $\cdot; e \rightarrow^{p*} H_1; v$ *and* $e \Downarrow_{AOT} M$ *then* $\cdot; M \rightarrow^{p*} H_2; v$

Subject Reduction Is Type Preservation

- Type Preservation is used for optimization, safety.

Proposition 2 (Type Preservation)
If $\Gamma \vdash e : A$ *and* $e \Downarrow_{AOT} M$ *then* $\Gamma' \vdash M : B$ *(ish)*

Subject Reduction Is Type Preservation

- Type Preservation is used for optimization, safety.
- Subject reduction is a standard lemma about reduction.

Proposition 2 (Type Preservation)
If $\Gamma \vdash e : A$ *and* $e \Downarrow_{AOT} M$ *then* $\Gamma' \vdash M : B$ *(ish)*

Proposition 3 (Subject Reduction)
If $\Gamma \vdash e : A$ *and* $H; e \to^{p} H'; e'$ *then* $\Gamma \vdash e' : A$

Subject Reduction Is Type Preservation

- Type Preservation is used for optimization, safety.
- Subject reduction is a standard lemma about reduction.
- Subject reduction implies type preservation, pretty trivially.

Proposition 2 (Type Preservation)
If $\Gamma \vdash e : A$ *and* $e \Downarrow_{AOT} M$ *then* $\Gamma' \vdash M : B$ *(ish)*

Proposition 3 (Subject Reduction)
If $\Gamma \vdash e : A$ *and* $H; e \rightarrow^{p} H'; e'$ *then* $\Gamma \vdash e' : A$

Theorem 1 (Subject Reduction Implies Type Safety)
If $(\Gamma \vdash e : A$ *and* $H; e \rightarrow^{p} H'; e'$ *implies* $\Gamma \vdash e' : A)$ *then*
$(\Gamma \vdash e : A$ *and* $e \Downarrow_{AOT} M$ *implies* $\Gamma' \vdash M : B)$

Subject Reduction Is Type Preservation

- Type Preservation is used for optimization, safety.
- Subject reduction is a standard lemma about reduction.
- Subject reduction implies type preservation, pretty trivially.

$$\boxed{e \Downarrow_{AOT} M}$$

$$\frac{AS(e) \ {}^{s}\!\rightarrow^{a}\!* \ M \qquad M \ {}^{s}\!\not\rightarrow^{a} \ M'}{e \Downarrow_{AOT} M}$$

Proposition 2 (Type Preservation)
If $\Gamma \vdash e : A$ *and* $e \Downarrow_{AOT} M$ *then* $\Gamma' \vdash M : B$ *(ish)*

Proposition 3 (Subject Reduction)
If $\Gamma \vdash e : A$ *and* $H; e \rightarrow^{p} H'; e'$ *then* $\Gamma \vdash e' : A$

Theorem 1 (Subject Reduction Implies Type Safety)
If $(\Gamma \vdash e : A$ *and* $H; e \rightarrow^{p} H'; e'$ *implies* $\Gamma \vdash e' : A)$ *then*
$(\Gamma \vdash e : A$ *and* $e \Downarrow_{AOT} M$ *implies* $\Gamma' \vdash M : B)$

Easy half of Full Abstraction is Trivial, by confluence

- Full abstraction is useful in secure compilation

Definition 1 (Contextual Equivalence)

$$
\begin{array}{lcll}
H_1; p_1 & \approx_{SA} & H_2; p_2 & \stackrel{\text{def}}{=} \forall C. H_1; C[p_1] \Downarrow^p \text{iff} H_2; C[p_2] \Downarrow^p \\
H_1; e_1 & \approx_{S} & H_2; e_2 & \stackrel{\text{def}}{=} \forall C_S. H_1; C_S[e_1] \Downarrow^s \text{iff} H_2; C_S[e_2] \Downarrow^s \\
H_1; M_1 & \approx_{A} & H_2; M_2 & \stackrel{\text{def}}{=} \forall C_A. H_1; C_A[M_1] \Downarrow^a \text{iff} H_2; C_A[M_2] \Downarrow^a
\end{array}
$$

Proposition 4 (Full Abstraction)
Suppose $e_1 \Downarrow_{AOT} M_1$ *and* $e_2 \Downarrow_{AOT} M_2$.
$H_1; e_1 \approx_S H_2; e_2$ *if and only if* $H_1; M_1 \approx_A H_2; M_2$.

Easy half of Full Abstraction is Trivial, by confluence

- Full abstraction is useful in secure compilation
  - Indistinguishable source compiled to indistinguishable target terms.
  - Target “attackers” can’t learn anything new.

Definition 1 (Contextual Equivalence)

$$
\begin{array}{lcll}
H_1; p_1 & \approx_{SA} & H_2; p_2 & \stackrel{\text{def}}{=} \forall C. H_1; C[p_1] \Downarrow^p \text{iff} H_2; C[p_2] \Downarrow^p \\
H_1; e_1 & \approx_{S} & H_2; e_2 & \stackrel{\text{def}}{=} \forall C_S. H_1; C_S[e_1] \Downarrow^s \text{iff} H_2; C_S[e_2] \Downarrow^s \\
H_1; M_1 & \approx_{A} & H_2; M_2 & \stackrel{\text{def}}{=} \forall C_A. H_1; C_A[M_1] \Downarrow^a \text{iff} H_2; C_A[M_2] \Downarrow^a
\end{array}
$$

Proposition 4 (Full Abstraction)
Suppose $e_1 \Downarrow_{AOT} M_1$ *and* $e_2 \Downarrow_{AOT} M_2$.
$H_1; e_1 \approx_S H_2; e_2$ *if and only if* $H_1; M_1 \approx_A H_2; M_2$.

Easy half of Full Abstraction is Trivial, by confluence

- Full abstraction is useful in secure compilation
  - Indistinguishable source compiled to indistinguishable target terms.
  - Target “attackers” can’t learn anything new.
- Multi-language: splits into two lemmas

Definition 1 (Contextual Equivalence)

$$
\begin{array}{lcll}
H_1; p_1 & \approx_{SA} & H_2; p_2 & \stackrel{\text{def}}{=} \forall C. H_1; C[p_1] \Downarrow^p \text{iff} H_2; C[p_2] \Downarrow^p \\
H_1; e_1 & \approx_{S} & H_2; e_2 & \stackrel{\text{def}}{=} \forall C_S. H_1; C_S[e_1] \Downarrow^s \text{iff} H_2; C_S[e_2] \Downarrow^s \\
H_1; M_1 & \approx_{A} & H_2; M_2 & \stackrel{\text{def}}{=} \forall C_A. H_1; C_A[M_1] \Downarrow^a \text{iff} H_2; C_A[M_2] \Downarrow^a
\end{array}
$$

Proposition 4 (Full Abstraction)
Suppose $e_1 \Downarrow_{AOT} M_1$ *and* $e_2 \Downarrow_{AOT} M_2$.
$H_1; e_1 \approx_S H_2; e_2$ *if and only if* $H_1; M_1 \approx_A H_2; M_2$.

Easy half of Full Abstraction is Trivial, by confluence

- Full abstraction is useful in secure compilation
  - Indistinguishable source compiled to indistinguishable target terms.
  - Target “attackers” can’t learn anything new.
- Multi-language: splits into two lemmas
  - Multi-language attackers can’t learn anything new, compared to source.

Definition 1 (Contextual Equivalence)

$$
\begin{array}{lcll}
H_1; p_1 & \approx_{SA} & H_2; p_2 & \stackrel{\text{def}}{=} \forall C. H_1; C[p_1] \Downarrow^p \text{iff } H_2; C[p_2] \Downarrow^p \\
H_1; e_1 & \approx_{S} & H_2; e_2 & \stackrel{\text{def}}{=} \forall C_S. H_1; C_S[e_1] \Downarrow^s \text{iff } H_2; C_S[e_2] \Downarrow^s \\
H_1; M_1 & \approx_{A} & H_2; M_2 & \stackrel{\text{def}}{=} \forall C_A. H_1; C_A[M_1] \Downarrow^a \text{iff } H_2; C_A[M_2] \Downarrow^a
\end{array}
$$

Proposition 4 (Full Abstraction)
Suppose $e_1 \Downarrow_{AOT} M_1$ *and* $e_2 \Downarrow_{AOT} M_2$.
$H_1; e_1 \approx_S H_2; e_2$ *if and only if* $H_1; M_1 \approx_A H_2; M_2$.

Lemma 1 (Full Abstraction (hard part))
Suppose $e_1 \Downarrow_{AOT} M_1$ *and* $e_2 \Downarrow_{AOT} M_2$.
$H_1; e_1 \approx_S H_2; e_2$ *if and only if* $H_1; M_1 \approx_{SA} H_2; M_2$.

Easy half of Full Abstraction is Trivial, by confluence

- Full abstraction is useful in secure compilation
  - Indistinguishable source compiled to indistinguishable target terms.
  - Target “attackers” can’t learn anything new.
- Multi-language: splits into two lemmas
  - Multi-language attackers can’t learn anything new, compared to source.
  - Target attackers can’t learn anything new, compared to multi-language

Definition 1 (Contextual Equivalence)

$$
\begin{array}{lcll}
H_1; p_1 & \approx_{SA} & H_2; p_2 & \stackrel{\text{def}}{=} \forall C. H_1; C[p_1] \Downarrow^p \text{iff} H_2; C[p_2] \Downarrow^p \\
H_1; e_1 & \approx_{S} & H_2; e_2 & \stackrel{\text{def}}{=} \forall C_S. H_1; C_S[e_1] \Downarrow^s \text{iff} H_2; C_S[e_2] \Downarrow^s \\
H_1; M_1 & \approx_{A} & H_2; M_2 & \stackrel{\text{def}}{=} \forall C_A. H_1; C_A[M_1] \Downarrow^a \text{iff} H_2; C_A[M_2] \Downarrow^a
\end{array}
$$

Proposition 4 (Full Abstraction)
Suppose $e_1 \Downarrow_{AOT} M_1$ *and* $e_2 \Downarrow_{AOT} M_2$.
$H_1; e_1 \approx_S H_2; e_2$ *if and only if* $H_1; M_1 \approx_A H_2; M_2$.

Lemma 1 (Full Abstraction (hard part))
Suppose $e_1 \Downarrow_{AOT} M_1$ *and* $e_2 \Downarrow_{AOT} M_2$.
$H_1; e_1 \approx_S H_2; e_2$ *if and only if* $H_1; M_1 \approx_{SA} H_2; M_2$.

Lemma 2 (Full Abstraction (easy part))
Suppose $e_1 \Downarrow_{AOT} M_1$ *and* $e_2 \Downarrow_{AOT} M_2$.
$H_1; e_1 \approx_{SA} H_2; e_2$ *if and only if* $H_1; M_1 \approx_A H_2; M_2$.

Full Abstraction Proof

Definition 1 (Contextual Equivalence)
$H_1; e_1 \approx H_2; e_2 \stackrel{\text{def}}{=} \forall C. H_1; C[e_1] \Downarrow^p \text{iff} H_2; C[e_2] \Downarrow^p$

Lemma 2 (Full Abstraction (easy part))
Suppose $e_1 \Downarrow_{AOT} M_1$ *and* $e_2 \Downarrow_{AOT} M_2$*.*
$H_1; e_1 \approx_{SA} H_2; e_2$ *if and only if* $H_1; M_1 \approx_A H_2; M_2$*.*

Full Abstraction Proof

- e1 -> M1, and e1 -> p1; by confluence M1 -> p1
- C[p1] either diverges, or doesn't.

Definition 1 (Contextual Equivalence)

$$H_1; e_1 \approx H_2; e_2 \stackrel{\text{def}}{=} \forall C. H_1; C[e_1] \Downarrow^p \text{iff} H_2; C[e_2] \Downarrow^p$$

Lemma 2 (Full Abstraction (easy part))

Suppose $e_1 \Downarrow_{AOT} M_1$ *and* $e_2 \Downarrow_{AOT} M_2$.
$H_1; e_1 \approx_{SA} H_2; e_2$ *if and only if* $H_1; M_1 \approx_A H_2; M_2$.

Full Abstraction Proof

- e1 -> M1, and e1 -> p1; by confluence M1 -> p1
- C[p1] either diverges, or doesn't.

- e1 -> M2, and e2 -> p2; by confluence, M1 -> p2.
- C[p2] either diverges or doesn't.

Definition 1 (Contextual Equivalence)

$$H_1; e_1 \approx H_2; e_2 \stackrel{\text{def}}{=} \forall C. H_1; C[e_1] \Downarrow^p \text{iff} H_2; C[e_2] \Downarrow^p$$

Lemma 2 (Full Abstraction (easy part))

Suppose $e_1 \Downarrow_{AOT} M_1$ *and* $e_2 \Downarrow_{AOT} M_2$.
$H_1; e_1 \approx_{SA} H_2; e_2$ *if and only if* $H_1; M_1 \approx_A H_2; M_2$.

Full Abstraction Proof

- e1 -> M1, and e1 -> p1; by confluence M1 -> p1
- C[p1] either diverges, or doesn't.

- e1 -> M2, and e2 -> p2; by confluence, M1 -> p2.
- C[p2] either diverges or doesn't.

- Since e1 ~ e2, C[p2] diverges iff C[p1] diverges

Definition 1 (Contextual Equivalence)

$$H_1; e_1 \approx H_2; e_2 \stackrel{\text{def}}{=} \forall C. H_1; C[e_1] \Downarrow^p \text{iff} H_2; C[e_2] \Downarrow^p$$

Lemma 2 (Full Abstraction (easy part))

Suppose $e_1 \Downarrow_{AOT} M_1$ *and* $e_2 \Downarrow_{AOT} M_2$.
$H_1; e_1 \approx_{SA} H_2; e_2$ *if and only if* $H_1; M_1 \approx_A H_2; M_2$.

Full Abstraction Proof

- e1 -> M1, and e1 -> p1; by confluence M1 -> p1
- C[p1] either diverges, or doesn't.

- e1 -> M2, and e2 -> p2; by confluence, M1 -> p2.
- C[p2] either diverges or doesn't.

- Since e1 ~ e2, C[p2] diverges iff C[p1] diverges
- Hence M1 ~ M2

Definition 1 (Contextual Equivalence)

$$H_1; e_1 \approx H_2; e_2 \stackrel{\text{def}}{=} \forall C. H_1; C[e_1] \Downarrow^p \text{iff} H_2; C[e_2] \Downarrow^p$$

Lemma 2 (Full Abstraction (easy part))

Suppose $e_1 \Downarrow_{AOT} M_1$ *and* $e_2 \Downarrow_{AOT} M_2$.
$H_1; e_1 \approx_{SA} H_2; e_2$ *if and only if* $H_1; M_1 \approx_A H_2; M_2$.

Discussion

- + The technique seems to scale
- + Interesting corollaries and derived models
- - Confluence is not easy

Compilation as Multi-Language Semantics

- Define:
  - operational semantics for source and target
  - multi-language operational semantics of interoperability, over open terms
- Get:
  - An ahead-of-time compiler, by normalizing cross-language reduction
  - A just-in-time compiler, by using running in multi-language reduction
- Prove confluence:
  - Implies (compositional) correctness of compilers
  - Implies part of the full abstraction (secure compilation) proof