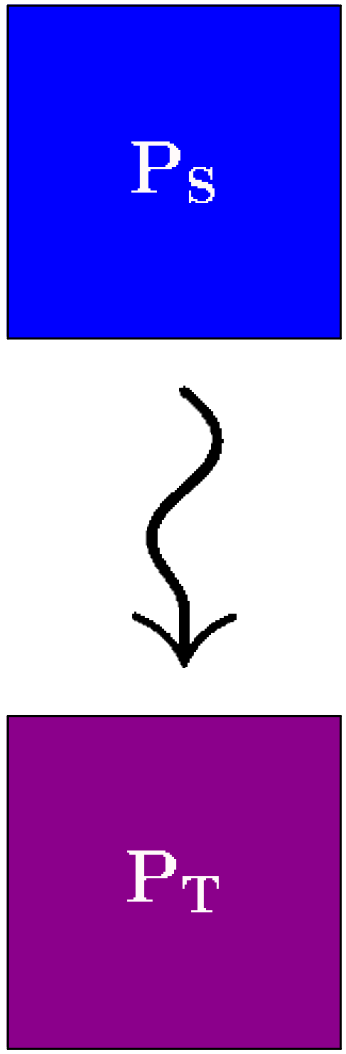
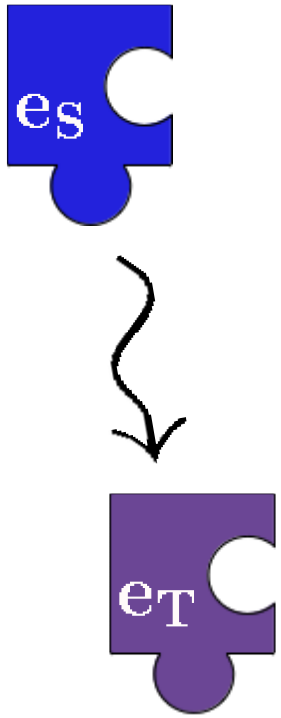
Verifying an Open Compiler from ML to Assembly

James T. Perconti Amal Ahmed

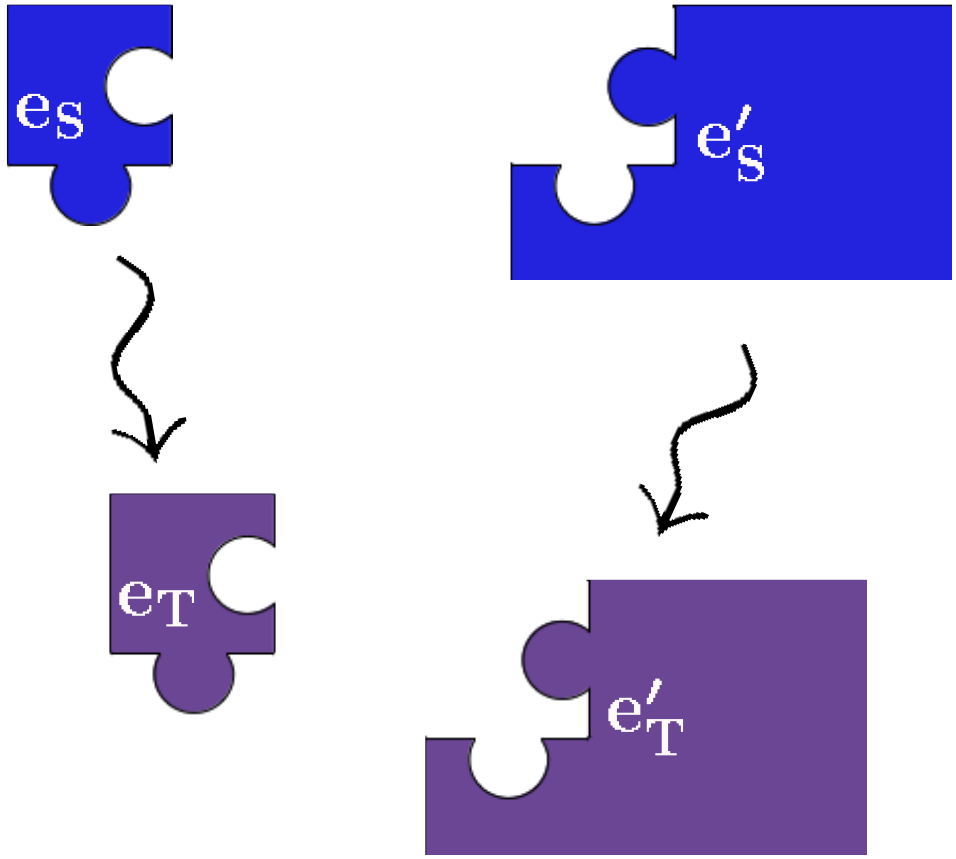
Whole-Program Compilers



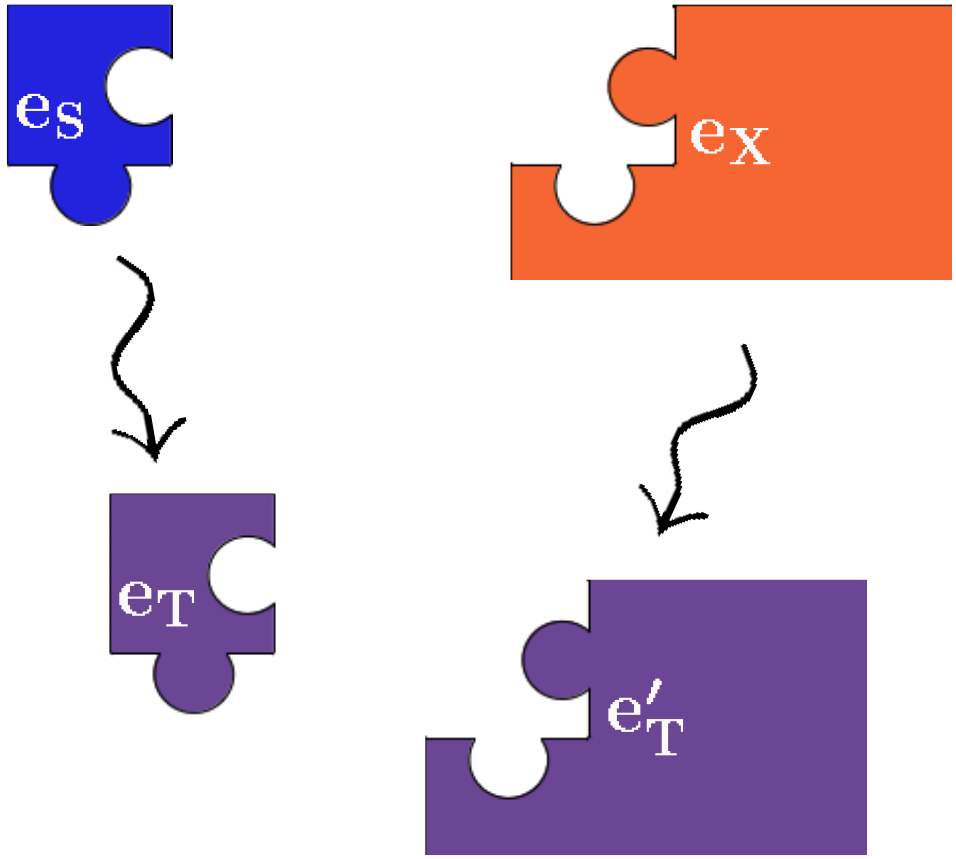
Open Compilers



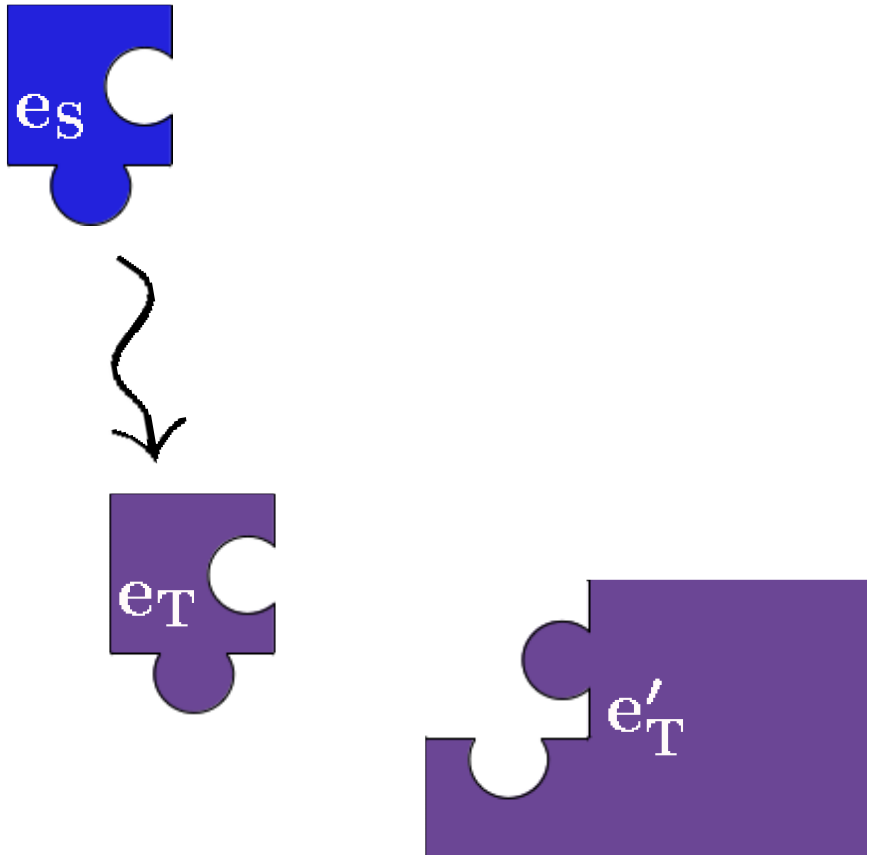
Open Compilers



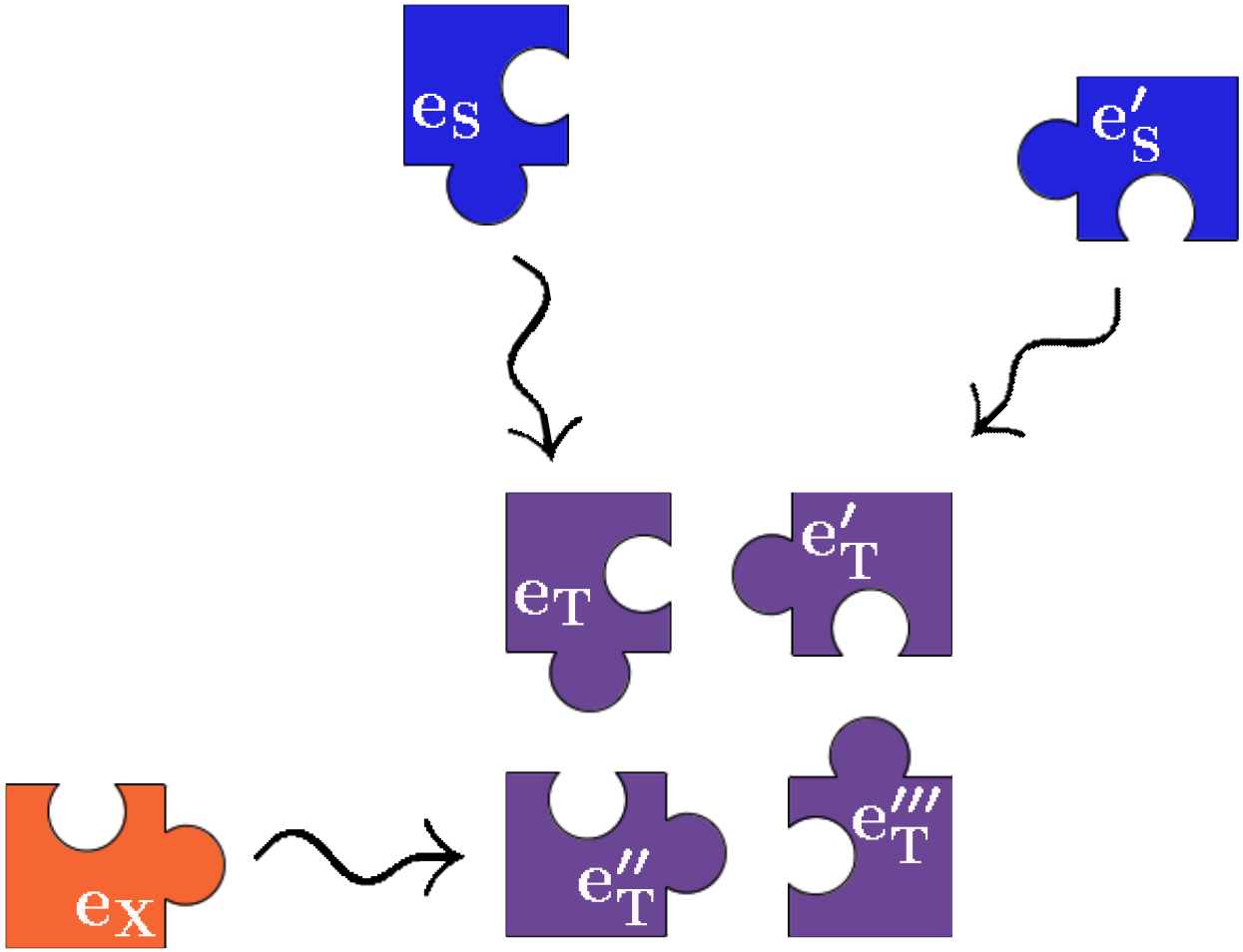
Open Compilers



Open Compilers



Open Compilers



Why Can't We Verify Open Compilers?

- Current techniques need closed-world
- Unclear correctness criterion

Key Question

What is the right methodology?

Outline

- Current state-of-the-art
- Our approach
- Challenges of realizing our approach

Current State-of-the-Art

$$\Gamma \vdash \mathbf{e_S} : \tau \rightsquigarrow \mathbf{e_T}$$

$$\Downarrow$$

$$\Gamma \vdash \mathbf{e_S} \approx \mathbf{e_T} : \tau$$

Benton-Hur ICFP '09

Hur-Dreyer POPL '11

Current State-of-the-Art

$\lambda f : int \rightarrow int.f\ 0$


```
add r2 r2 3
  ⋮
jmp r7
```

Current State-of-the-Art

$\lambda f : \mathrm{int} \to \mathrm{int}.f\ 0$

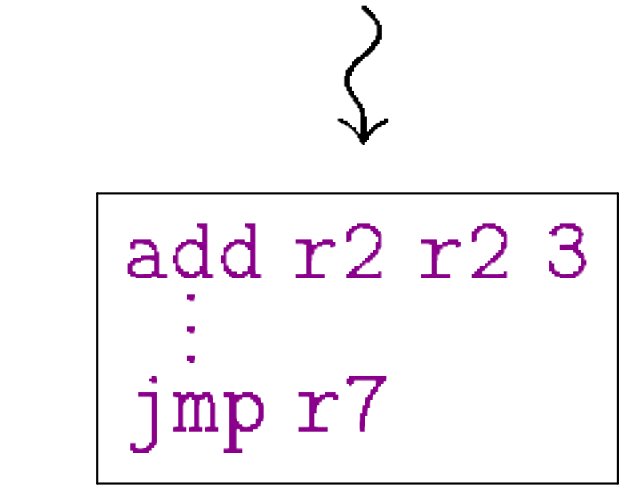
```
add r2 r2 3
  ⋮
jmp r7
```

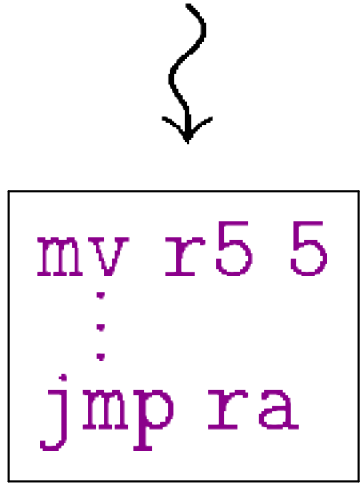
$\cdot \vdash \lambda f : \mathrm{int} \to \mathrm{int}.f\ 0 : (\mathrm{int} \to \mathrm{int}) \to \mathrm{int} \approx$

```
add r2 r2 3
  ⋮
jmp r7
```

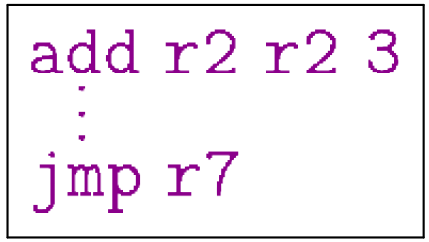
Current State-of-the-Art

$\lambda f : int \to int . f\ 0$ $\lambda x : int . 5$

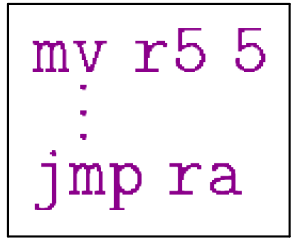




$\cdot \vdash \lambda f : int \to int . f\ 0 : (int \to int) \to int \approx$

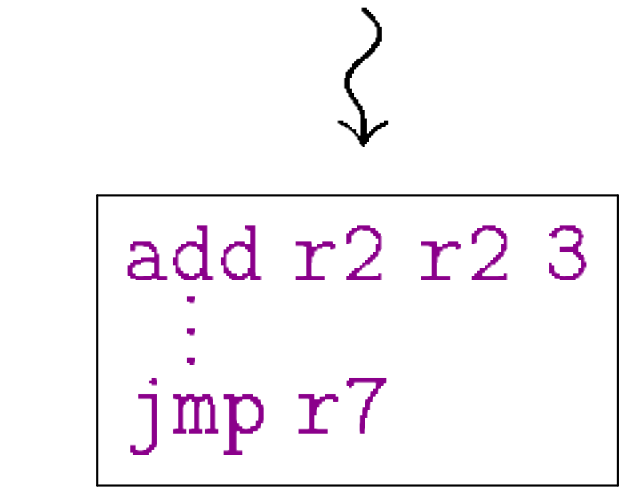


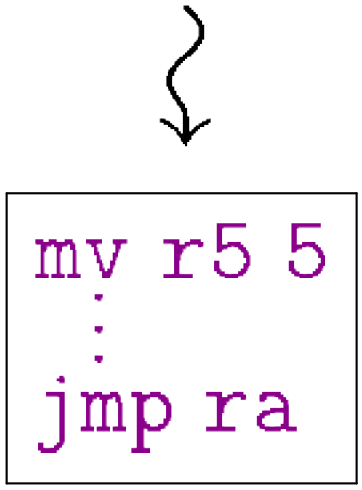
$\cdot \vdash \lambda x : int . 5 : int \to int \approx$



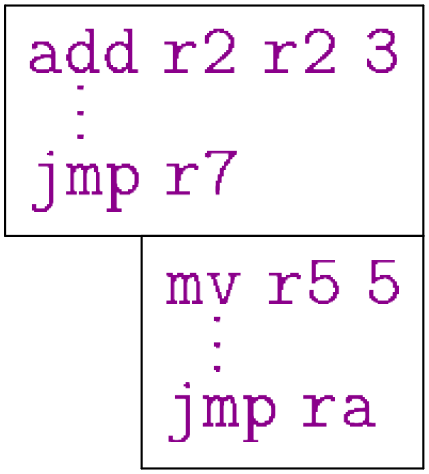
Current State-of-the-Art

$\lambda f : int \to int.f\ 0$ $\lambda x : int.5$



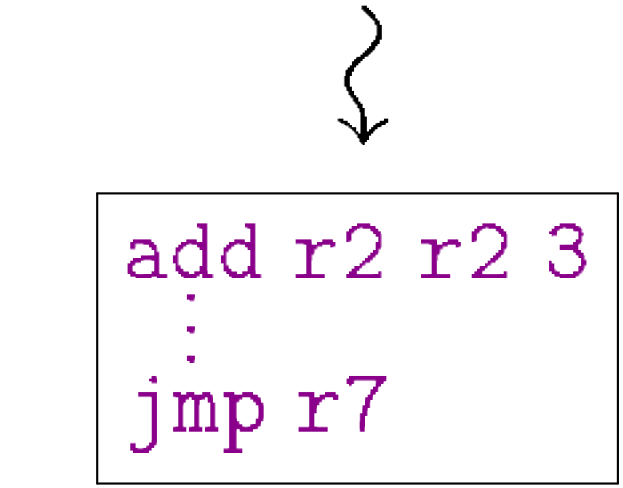


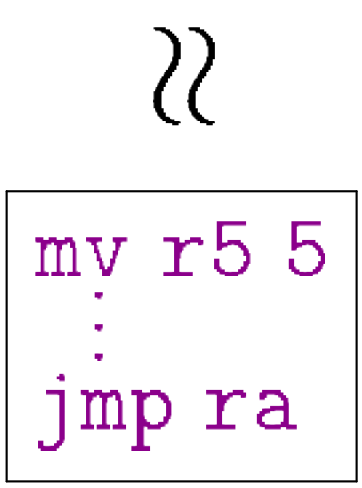
$\cdot \vdash (\lambda f : int \to int.f\ 0)\ (\lambda x : int.5) : int \approx$



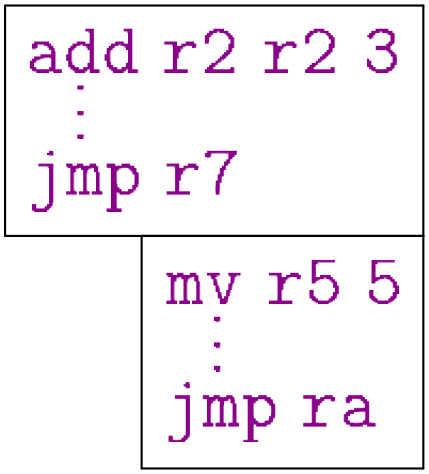
Current State-of-the-Art

$\lambda f : int \to int.f\ 0$ $\quad$ $\lambda x : int.5$





$\cdot \vdash (\lambda f : int \to int.f\ 0)\ (\lambda x : int.5) : int \approx$

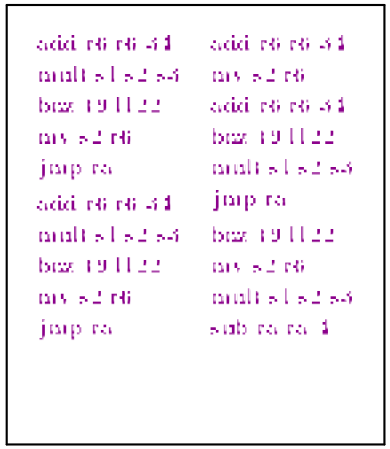


Current State-of-the-Art

$\lambda f : int \rightarrow int . f\ 0$


```
add r2 r2 3
   ⋮
jmp r7
```

?

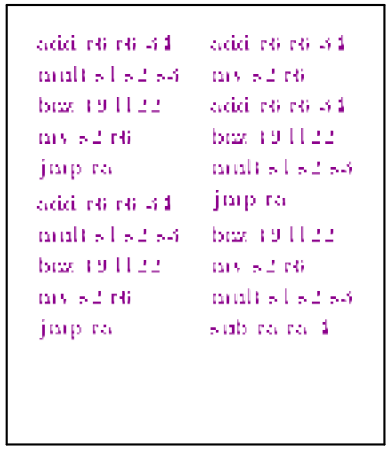


Current State-of-the-Art

$\lambda f : int \rightarrow int . f\ 0$


```
add r2 r2 3
   ⋮
jmp r7
```

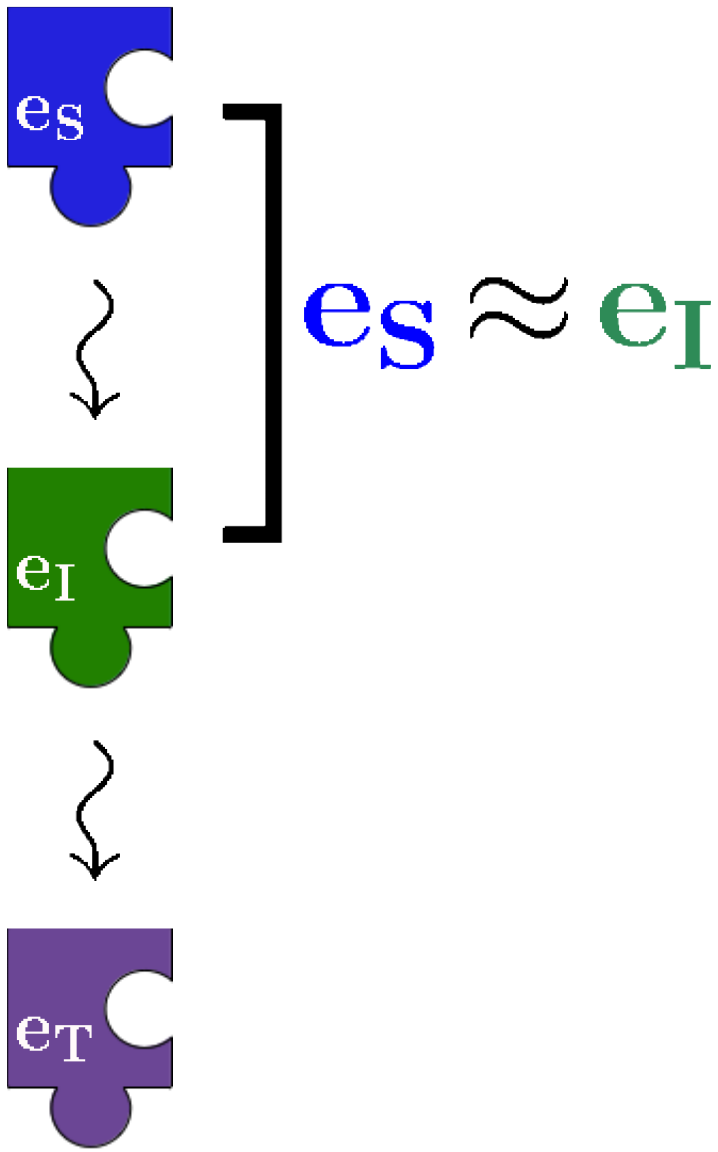
?



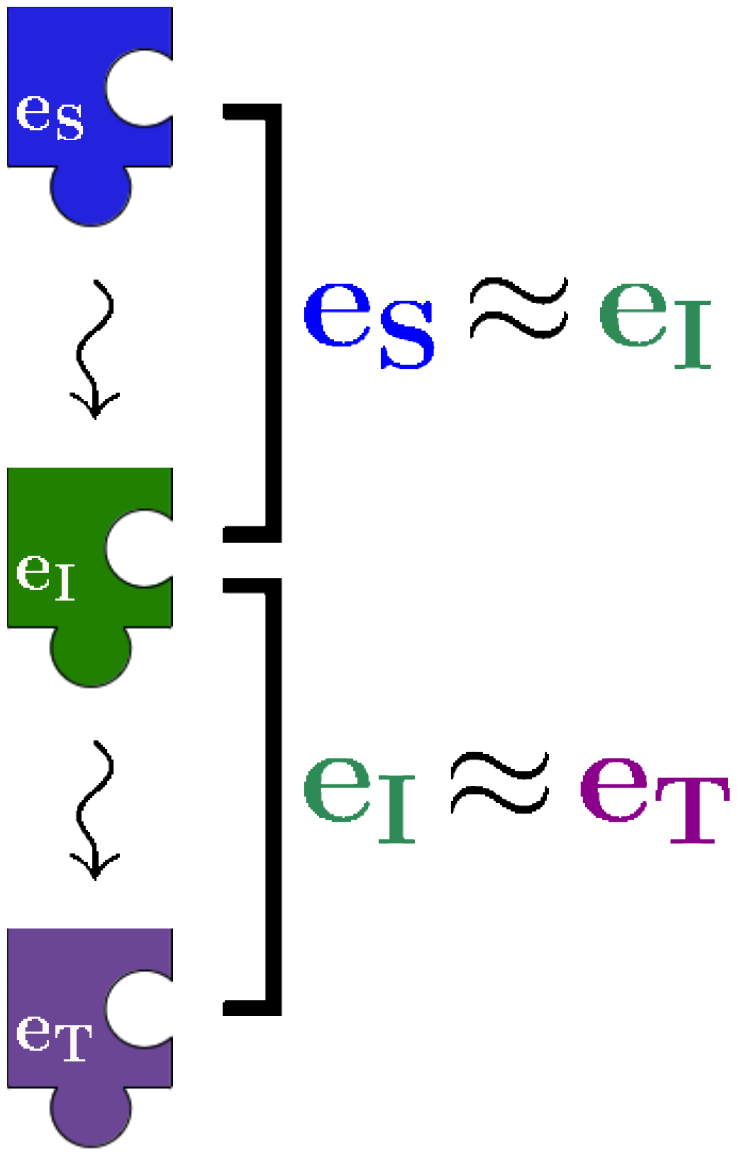
Limitation 1

Multi-Pass Compilers

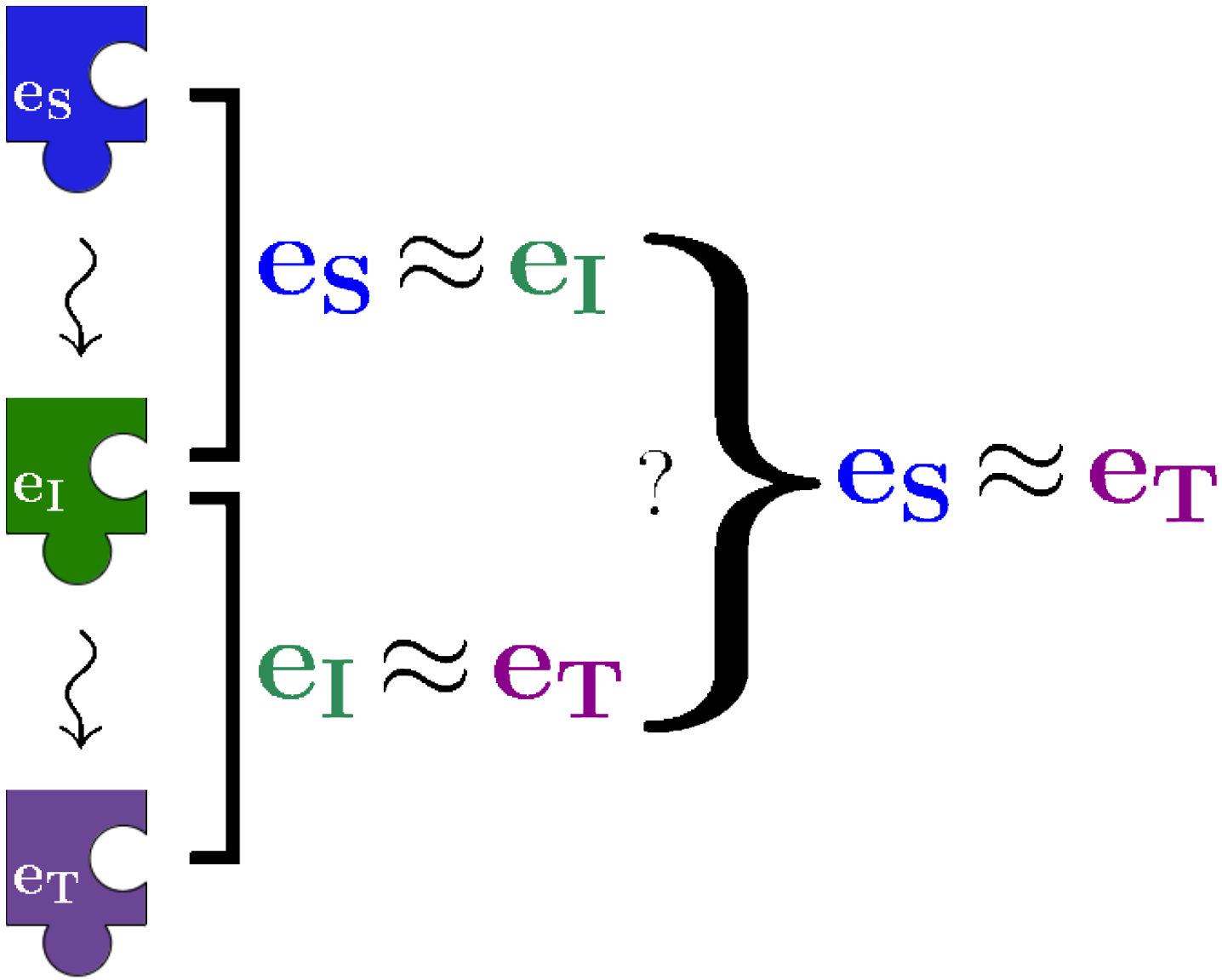
Multi-Pass Compilers



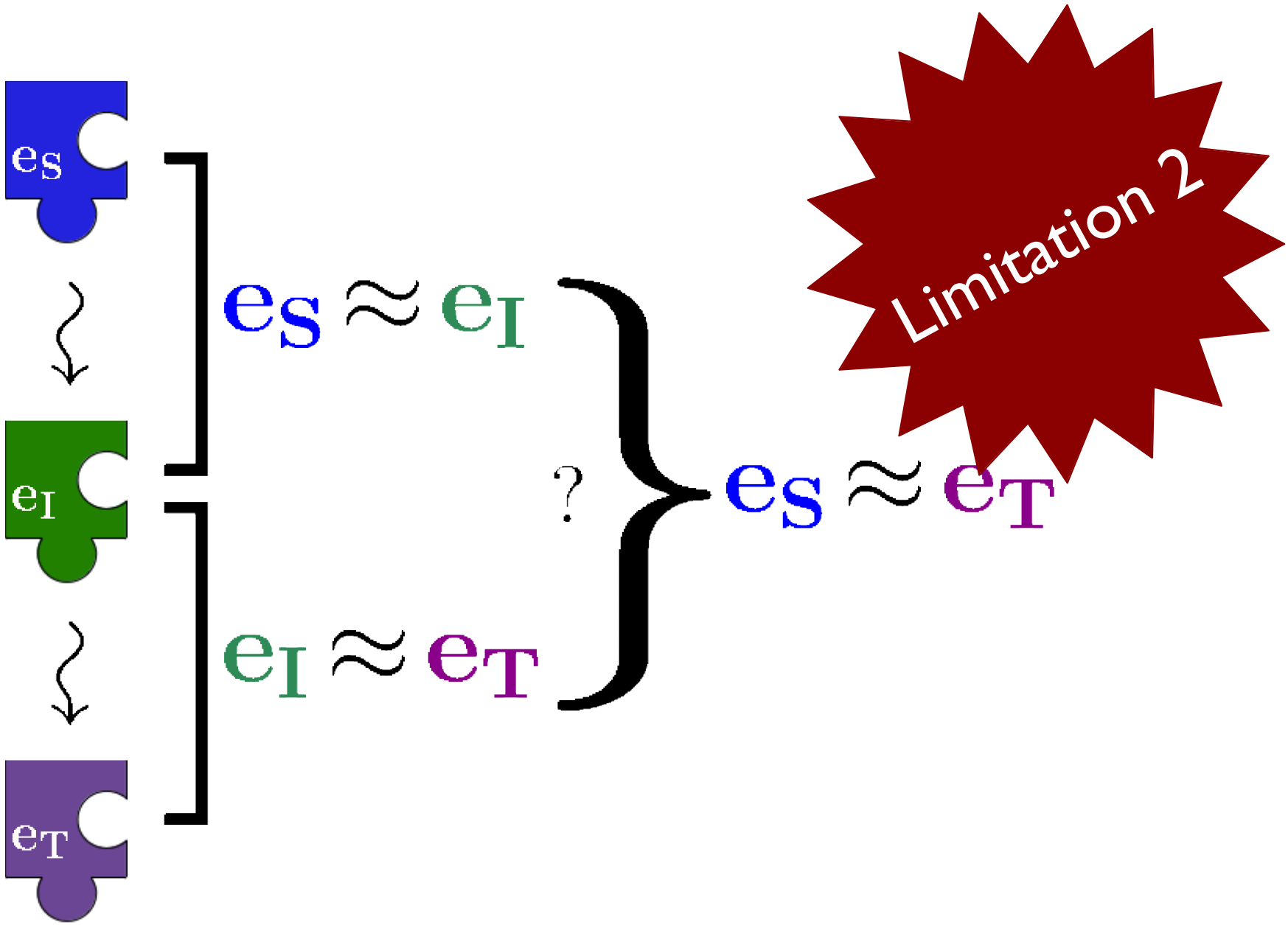
Multi-Pass Compilers



Multi-Pass Compilers



Multi-Pass Compilers



Limitations of Current Approach

1. Need to find related source component
2. Can't compose relations for multiple passes

Key Ideas

- Formalize semantics of interoperability

Key Ideas

- Formalize semantics of interoperability
  - Adjacent pairs of languages

$$S \longleftrightarrow I_1 \longleftrightarrow \cdots \longleftrightarrow T$$

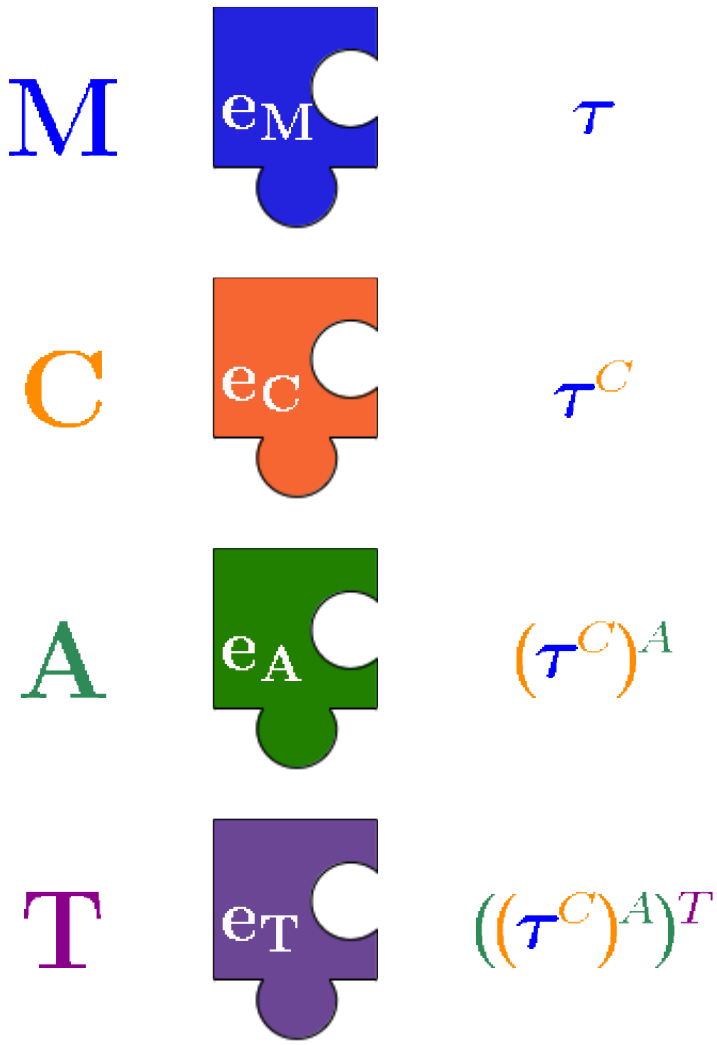
Key Ideas

- Formalize semantics of interoperability
  - Adjacent pairs of languages

$$S \longleftrightarrow I_1 \longleftrightarrow \cdots \longleftrightarrow T$$

- Challenge: high/low-level languages

MCAT



Interoperability Using Boundaries

$$^{\tau}\mathcal{MC}(\mathbf{e_C})$$

$$\mathcal{CM}^{\tau}(\mathbf{e_M})$$

Interoperability Using Boundaries

$$^{\tau}\mathcal{MC}(\mathbf{e_C}) \quad ^{\tau}\mathcal{CA}(\mathbf{e_A}) \quad ^{\tau}\mathcal{AT}(\mathbf{e_T})$$

$$\mathcal{CM}^{\tau}(\mathbf{e_M}) \quad \mathcal{AC}^{\tau}(\mathbf{e_C}) \quad \mathcal{TA}^{\tau}(\mathbf{e_A})$$

Semantics of Boundaries

$$^{\mathbf{int}}\mathcal{MC}(5) \longmapsto 5$$

Semantics of Boundaries

$$\tau_1 \to \tau_2 \mathcal{MC}(\mathbf{v_f})$$

Semantics of Boundaries

$$(\tau_1 \to \tau_2)^C = \tau_1{}^C \to \tau_2{}^C$$

$$\tau_1 \to \tau_2 \mathcal{MC}(\mathbf{v_f})$$

Semantics of Boundaries

$$(\tau_1 \to \tau_2)^C = {\tau_1}^C \to {\tau_2}^C$$

$$\tau_1 \to \tau_2 \mathcal{MC}(\mathbf{v_f}) \longmapsto \boldsymbol{\lambda} \mathbf{x}.^{\tau_2}\mathcal{MC}(\mathbf{v_f}\ \mathcal{CM}^{\tau_1}(\mathbf{x}))$$

Compiler Correctness Criterion

- Contextual Equivalence!

Compiler Correctness Criterion

- Contextual Equivalence!

$$\Gamma \vdash \mathbf{e_M} : \tau \rightsquigarrow \mathbf{e_C}$$

$$\Downarrow$$

$$\Gamma \vdash \mathbf{e_M} \approx^{\tau} \mathcal{MC}(\mathbf{e_C}) : \tau$$

Compiler Correctness Criterion

- Contextual Equivalence!

$$\Gamma \vdash \mathbf{e_M} : \tau \rightsquigarrow \mathbf{e_C}$$

$$\Downarrow$$

$$\Gamma \vdash \mathbf{e_M} \approx^{\tau} \mathcal{MC}(\mathbf{e_C}) : \tau$$

$$\Gamma \vdash \mathcal{CM}^{\tau}(\mathbf{e_M}) \approx \mathbf{e_C} : \tau^{C}$$

Compiler Correctness Criterion

$$\Gamma \vdash \mathbf{e_M} : \tau \rightsquigarrow \mathbf{e_C} \implies \Gamma \vdash \mathbf{e_M} \approx^{\tau} \mathcal{MC}(\mathbf{e_C}) : \tau$$

$$\Gamma \vdash \mathbf{e_C} : \tau \rightsquigarrow \mathbf{e_A} \implies \Gamma \vdash \mathbf{e_C} \approx^{\tau} \mathcal{CA}(\mathbf{e_A}) : \tau$$

$$\Gamma \vdash \mathbf{e_A} : \tau \rightsquigarrow \mathbf{e_T} \implies \Gamma \vdash \mathbf{e_A} \approx^{\tau} \mathcal{AT}(\mathbf{e_T}) : \tau$$

Compiler Correctness Criterion

$$\Gamma \vdash \mathbf{e_M} : \tau \rightsquigarrow \mathbf{e_C} \implies \Gamma \vdash \mathbf{e_M} \approx^{\tau} \mathcal{MC}(\mathbf{e_C}) : \tau$$

$$\Gamma \vdash \mathbf{e_C} : \tau \rightsquigarrow \mathbf{e_A} \implies \Gamma \vdash \mathbf{e_C} \approx^{\tau} \mathcal{CA}(\mathbf{e_A}) : \tau$$

$$\Gamma \vdash \mathbf{e_A} : \tau \rightsquigarrow \mathbf{e_T} \implies \Gamma \vdash \mathbf{e_A} \approx^{\tau} \mathcal{AT}(\mathbf{e_T}) : \tau$$

$$\Gamma \vdash \mathbf{e_M} : \tau \rightsquigarrow \mathbf{e_T} \implies \Gamma \vdash \mathbf{e_M} \approx^{\tau} \mathcal{MC}(\tau^{\mathcal{C}} \mathcal{CA}(\tau^{\mathcal{CA}} \mathcal{AT}(\mathbf{e_T}))) : \tau$$

Compiler Correctness Criterion

$$\Gamma \vdash \mathbf{e_M} : \tau \rightsquigarrow \mathbf{e_C} \implies \Gamma \vdash \mathbf{e_M} \approx^{\tau} \mathcal{MC}(\mathbf{e_C}) : \tau$$

$$\Gamma \vdash \mathbf{e_C} : \tau \rightsquigarrow \mathbf{e_A} \implies \Gamma \vdash \mathbf{e_C} \approx^{\tau} \mathcal{CA}(\mathbf{e_A}) : \tau$$

$$\Gamma \vdash \mathbf{e_A} : \tau \rightsquigarrow \mathbf{e_T} \implies \Gamma \vdash \mathbf{e_A} \approx^{\tau} \mathcal{AT}(\mathbf{e_T}) : \tau$$

$$\Gamma \vdash \mathbf{e_M} : \tau \rightsquigarrow \mathbf{e_T} \implies \Gamma \vdash \mathbf{e_M} \approx^{\tau} \mathcal{MCAT}(\mathbf{e_T}) : \tau$$

Finding a Source Component

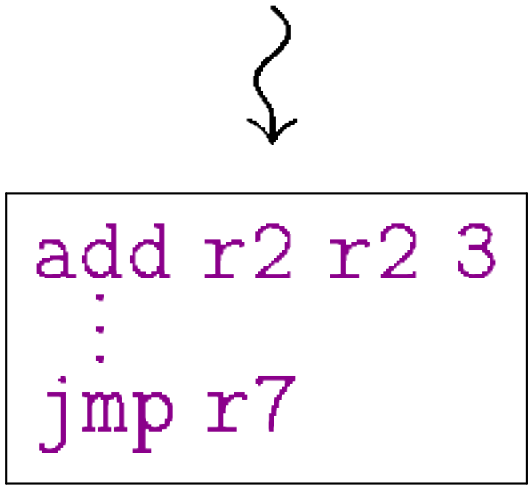
$\lambda f : int \rightarrow int . f\ 0$?

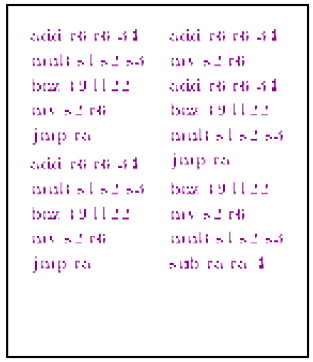
```
add r2 r2 3
   ⋮
jmp r7
```

```
addi r6 r6 -4 |  addi r6 r6 -4
muli s1 s2 s4 |  mv  s2 r6
bnz  r9 l122  |  addi r6 r6 -4
mv  s2 r6     |  bnz  r9 l122
jmp ra        |  muli s1 s2 s4
addi r6 r6 -4 |  jmp ra
muli s1 s2 s4 |  bnz  r9 l122
bnz  r9 l122  |  mv  s2 r6
mv  s2 r6     |  muli s1 s2 s4
jmp ra        |  sub ra ra 4
```

Finding a Source Component

$\lambda f : \mathrm{int} \to \mathrm{int}.f\ 0$?

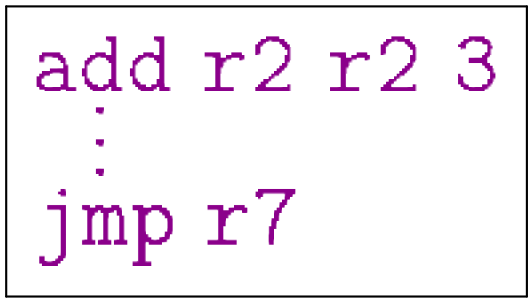


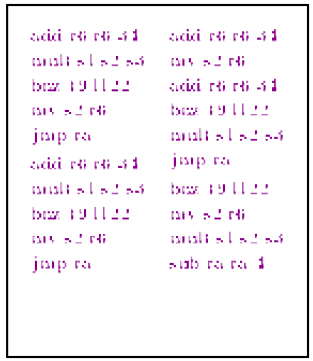


$\cdot \vdash \mathcal{TCAM}^{\mathrm{int}}((\lambda f : \mathrm{int} \to \mathrm{int}.f\ 0)) : \mathrm{int} \approx$

Finding a Source Component

$\lambda f : \mathbf{int} \to \mathbf{int}.f\ 0$?





$\cdot \vdash \mathcal{TCAM}^{\mathbf{int}}((\lambda f : \mathbf{int} \to \mathbf{int}.f\ 0)^{\mathbf{int} \to \mathbf{int}} \mathcal{MCAT}$  $) : \mathbf{int} \approx$ 

Finding a Source Component

$$\Gamma \vdash \mathbf{e_M} : \tau' \to \tau \rightsquigarrow \mathbf{e_T}$$

$$\Downarrow$$

$$\Gamma \vdash \mathcal{TCAM}^{\tau}(\mathbf{e_M}\ (\tau' \mathcal{MCAT}(\mathbf{e'_T}))) \approx \mathbf{e_T}\ \mathbf{e'_T} : ((\tau^{C})^{A})^{T}$$

Success!

1. Can simulate related source component
2. Handles multiple passes

Challenges

1. Assembly/high-level language interoperation

Assembly is Different

$$\Gamma \vdash \mathbf{e_A} : \tau \rightsquigarrow \mathbf{e_T}$$

$$\Downarrow$$

$$\Gamma \vdash \mathbf{e_A} \approx^{\tau} \mathcal{AT}(\mathbf{e_T}) : \tau$$

Assembly is Different

$$\Gamma \vdash \mathbf{e_A} : \tau \rightsquigarrow (\mathbf{I}, \mathbf{H})$$

$$\Downarrow$$

$$\Gamma \vdash \mathbf{e_A} \approx^{\tau} \mathcal{AT}((\mathbf{I}, \mathbf{H})) : \tau$$

Challenges

1. Assembly/high-level language interoperation
2. Type abstraction
3. Mutable references
4. Allocation

Takeaways

- Want a guarantee for compiling components

Takeaways

- Want a guarantee for compiling components
- Need to find the right methodology

Takeaways

- Want a guarantee for compiling components
- Need to find the right methodology
- Multi-language semantics solves limitations of previous approach

Takeaways

- Want a guarantee for compiling components
- Need to find the right methodology
- Multi-language semantics solves limitations of previous approach
- Challenge: define interoperability for different kinds of languages