FunTAL: Reasonably Mixing a Functional Language with Assembly

Daniel Patterson,[*] Jamie Perconti,[*]
Christos Dimoulas,[†] Amal Ahmed[*]

June 20, 2017

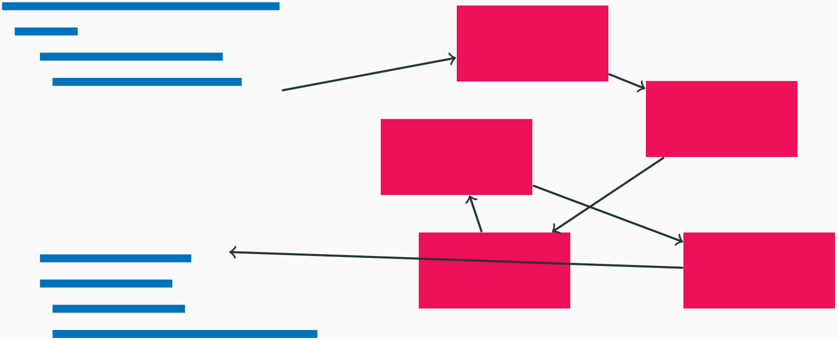
* Northeastern University, † Harvard University

Mixed language programs

Functional program

Mixed language programs

Functional program Inline assembly



Questions we want to answer

How to safely mix assembly with high-level code?

How to reason about equivalence of mixed programs?

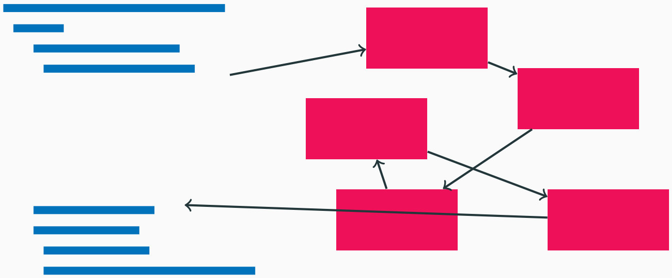
Questions we want to answer

How to safely mix assembly with high-level code?

How to reason about equivalence of mixed programs?

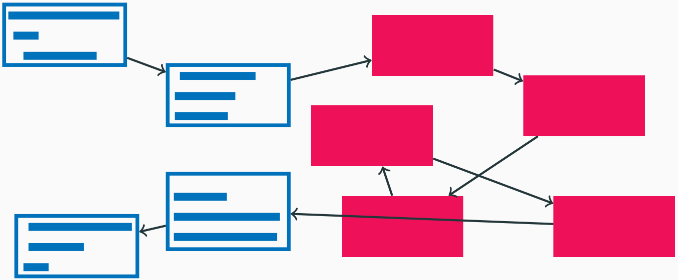
Prior approaches to mixing

Option 1: Translate high-level code into continuation-passing style to match assembly control-flow.



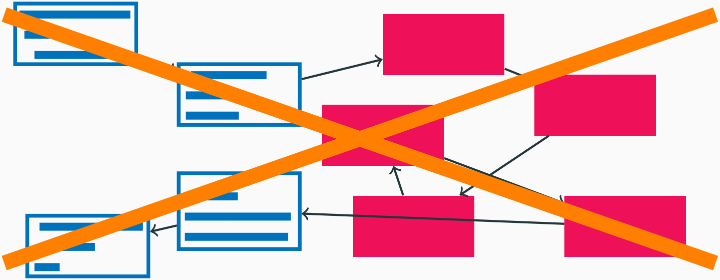
Prior approaches to mixing

Option 1: Translate high-level code into continuation-passing style to match assembly control-flow.



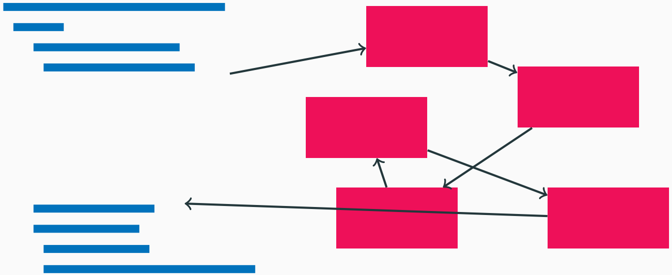
Prior approaches to mixing

Option 1: Translate high-level code into continuation-passing style to match assembly control-flow.



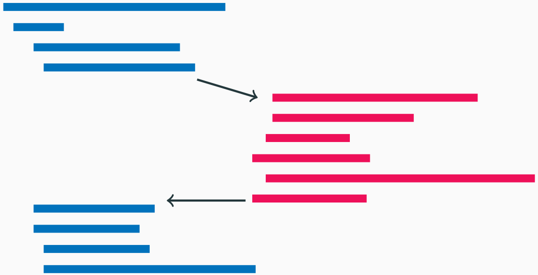
Prior approaches to mixing

Option 2: Impose high-level call-stack control-flow onto assembly.



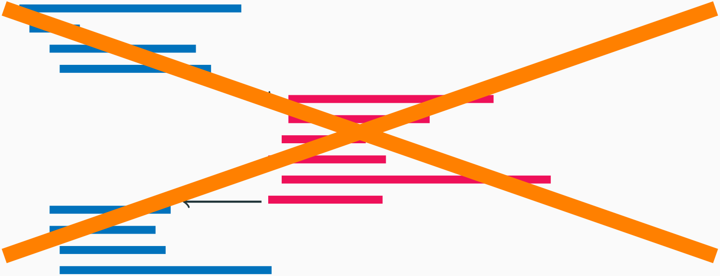
Prior approaches to mixing

Option 2: Impose high-level call-stack control-flow onto assembly.



Prior approaches to mixing

Option 2: Impose high-level call-stack control-flow onto assembly.



Our contributions

Allow safe mixing that allows **high-level code to remain high-level** and **low-level code to remain low-level**.

Our contributions

Allow safe mixing that allows **high-level code to remain high-level** and **low-level code to remain low-level**.

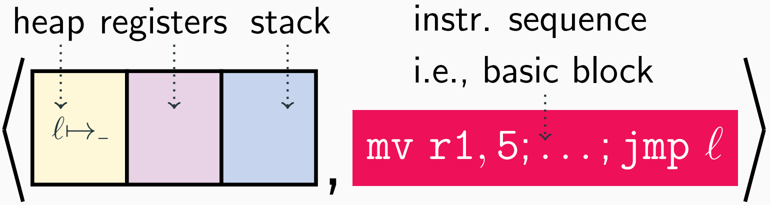
We do this via a novel notion of a **return marker**, which allows us to define the notion of an assembly **component**.

Fun: Functional language

- Simply typed lambda calculus (STLC)
- with (iso-)recursive types

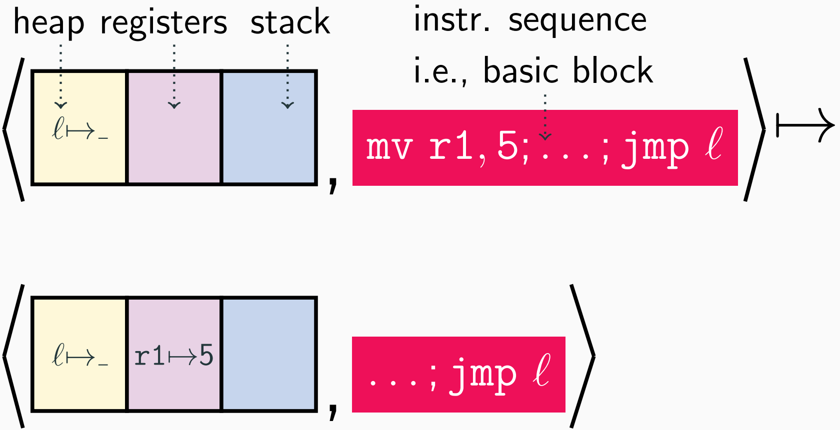
TAL: Typed Assembly Language

[Morrisett, Crary, Glew, Walker '98]



TAL: Typed Assembly Language

[Morrisett, Crary, Glew, Walker '98]



TAL: Typed Assembly Language

[Morrisett, Crary, Glew, Walker '98]

heap registers stack

instr. sequence

i.e., basic block

$$\left\langle \boxed{\ell \mapsto \_ \;\Big|\; \;\Big|\; } , \; \boxed{\texttt{mv r1}, 5; \ldots; \texttt{jmp}\ \ell} \right\rangle$$

$$\Psi\,;\Delta\,;\chi\,;\sigma \vdash \boxed{\texttt{instr}; \ldots; \texttt{jmp}\ \ell}$$

TAL: Typed Assembly Language

[Morrisett, Crary, Glew, Walker '98]

heap registers stack

instr. sequence

i.e., basic block

$$\left\langle \boxed{\ell \mapsto \_}\,\boxed{}\,\boxed{}\ ,\ \texttt{mv r1}, \texttt{5}; \ldots; \texttt{jmp}\ \ell \right\rangle$$

$$\Psi; \Delta; \chi; \sigma \vdash \texttt{instr}; \ldots; \texttt{jmp}\ \ell$$

heap typing

TAL: Typed Assembly Language

[Morrisett, Crary, Glew, Walker '98]

heap registers stack

instr. sequence

i.e., basic block

$\langle$ $\ell \mapsto \_$, $\texttt{mv r1}, 5; \ldots; \texttt{jmp } \ell$ $\rangle$

$\Psi; \Delta; \chi; \sigma \vdash \texttt{instr}; \ldots; \texttt{jmp } \ell$

type env

heap typing

TAL: Typed Assembly Language

[Morrisett, Crary, Glew, Walker '98]

heap registers stack

instr. sequence

i.e., basic block

$$\left\langle \boxed{\ell \mapsto \_}\ \boxed{}\ \boxed{}\ ,\ \boxed{\texttt{mv r1, 5; \ldots; jmp } \ell} \right\rangle$$

$$\Psi; \Delta; \chi; \sigma \vdash \boxed{\texttt{instr; \ldots; jmp } \ell}$$

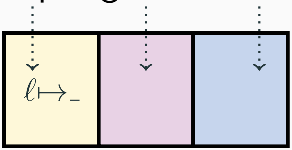
type env

register file typing

heap typing

TAL: Typed Assembly Language

[Morrisett, Crary, Glew, Walker '98]



instr. sequence
i.e., basic block

$$\langle\ \boxed{\quad}\boxed{\quad}\boxed{\quad}\ ,\ \texttt{mv r1}, 5; \ldots; \texttt{jmp}\ \ell\ \rangle$$

$$\Psi; \Delta; \chi; \sigma \vdash \texttt{instr}; \ldots; \texttt{jmp}\ \ell$$

heap typing

type env

register file typing

stack typing

TAL: Typed Assembly Language

[Morrisett, Crary, Glew, Walker '98]

heap registers stack

instr. sequence

i.e., basic block

$$\left\langle \boxed{\ell \mapsto \_ \;\Big|\; \;\Big|\; }\,,\ \boxed{\texttt{mv r1}, 5; \ldots; \texttt{jmp}\ \ell} \right\rangle$$

$$\Psi; \Delta; \chi; \sigma \vdash \boxed{\texttt{instr}; \ldots; \texttt{jmp}\ \ell}$$

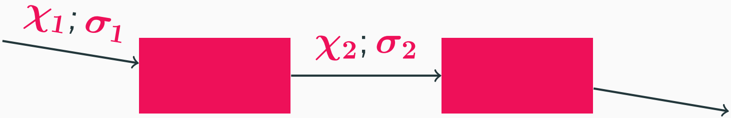
$$\boxed{\texttt{instr}; \ldots; \texttt{jmp}\ \ell} : \forall[\Delta].\{\chi; \sigma\}$$

TAL types are preconditions

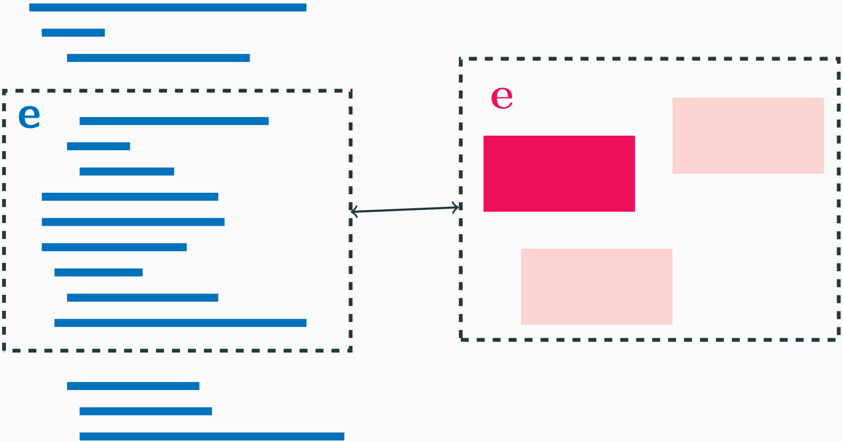
$$: \forall[\Delta].\{\chi; \sigma\}$$

TAL types are preconditions

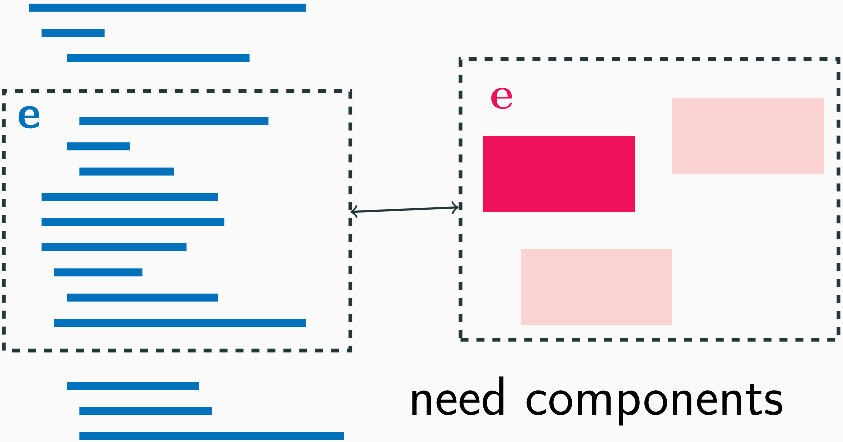
$$: \forall[\Delta].\{\chi; \sigma\}$$



Mixing is different

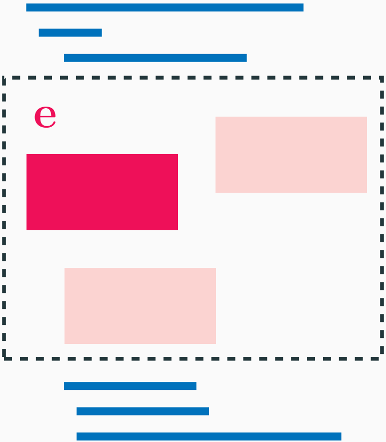


Mixing is different

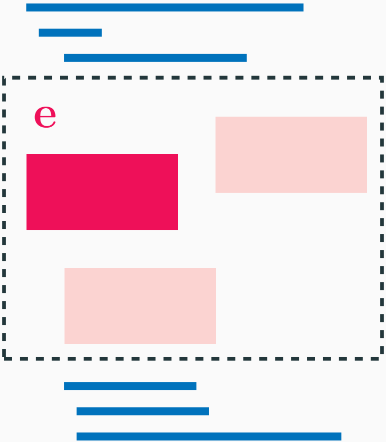


need components

Mixing is different

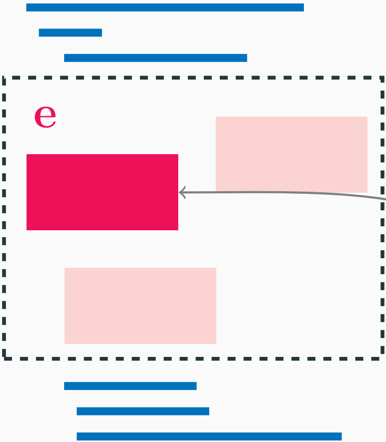


Mixing is different



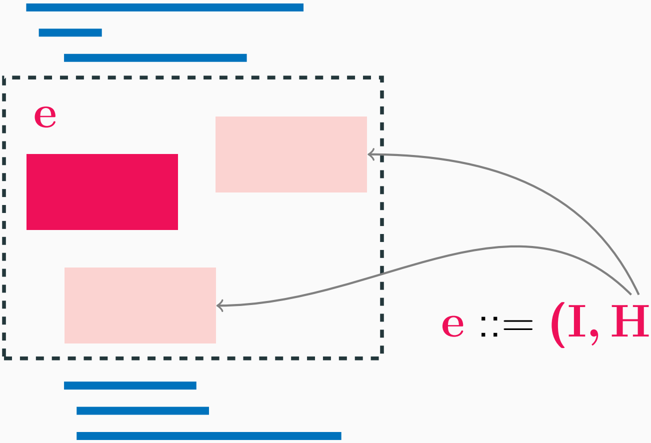
$$\mathbf{e} ::= (\mathbf{I}, \mathbf{H})$$

Mixing is different



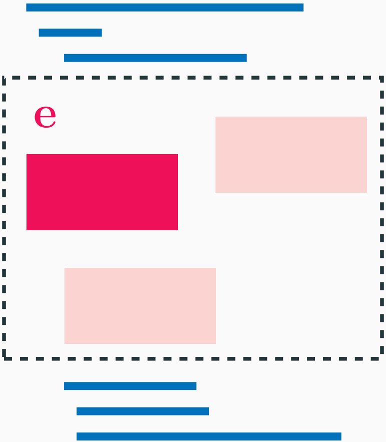
$$\mathbf{e} ::= (\mathbf{I}, \mathbf{H})$$

Mixing is different



$$\mathbf{e} ::= (\mathbf{I}, \mathbf{H})$$

Mixing is different



writing this program requires multi-language

Multi-languages in general

[Matthews-Findler '07]

Combine syntaxes from languages S and T and introduce boundary terms.

$$ {}^{\tau}ST(e_T) \mapsto^* {}^{\tau}ST(v_T) \mapsto v_S $$

$$ TS^{\tau}(e_S) \mapsto^* TS^{\tau}(v_S) \mapsto v_T $$

"Reduce under boundary and then translate value."

Multi-languages in general

[Matthews-Findler '07]

Combine syntaxes from languages S and T and introduce boundary terms.

$$^{\tau}ST(e_T) \mapsto^* \ ^{\tau}ST(v_T) \mapsto v_S$$

$$TS^{\tau}(e_S) \mapsto^* TS^{\tau}(v_S) \mapsto v_T$$

"Reduce under boundary and then translate value."

Multi-languages in general

[Matthews-Findler '07]

Combine syntaxes from languages S and T and introduce boundary terms.

$$^{\tau}ST(e_T) \mapsto^* \ {}^{\tau}ST(v_T) \mapsto v_S$$

$$TS^{\tau}(e_S) \mapsto^* TS^{\tau}(v_S) \mapsto v_T$$

"Reduce under boundary and then translate value."

Multi-languages in general

[Matthews-Findler '07]

Combine syntaxes from languages S and T and introduce boundary terms.

$$ {}^{\tau}ST(e_T) \mapsto^{*} {}^{\tau}ST(v_T) \mapsto v_S $$

$$ TS^{\tau}(e_S) \mapsto^{*} TS^{\tau}(v_S) \mapsto v_T $$

"Reduce under boundary and then translate value."

Multi-languages in general

[Matthews-Findler '07]

Boundary translations rely on a cross-language type translation $(\cdot)^+$.

$$\frac{e_S : \tau}{TS^{\tau}(e_S) : \tau^+} \qquad \frac{e_T : \tau^+}{{}^{\tau}ST(e_T) : \tau}$$

Multi-languages in general

[Matthews-Findler '07]

Boundary translations rely on a cross-language type translation $(\cdot)^+$.

$$\frac{e_S : \tau}{TS^{\tau}(e_S) : \tau^+} \qquad \frac{e_T : \tau^+}{{}^{\tau}ST(e_T) : \tau}$$

Multi-languages in general

[Matthews-Findler '07]

Boundary translations rely on a cross-language type translation $(\cdot)^+$.

$$\frac{e_S : \tau}{TS^{\tau}(e_S) : \tau^+} \qquad \frac{e_T : \tau^+}{{}^{\tau}ST(e_T) : \tau}$$

Multi-language for FunTAL

$$\boldsymbol{\tau}\mathcal{FT}(\mathbf{I}, \mathbf{H}) \longmapsto^{*} \boldsymbol{\tau}\mathcal{FT}(\mathbf{v}) \longmapsto \mathbf{v}$$

Multi-language for FunTAL

$$\tau\mathcal{FT}(\mathbf{I}, \mathbf{H}) \mapsto^{*} \tau\mathcal{FT}(\mathbf{v}) \mapsto \mathbf{v}$$

$$(\mathbf{I}, \mathbf{H}) : \mathbf{?}$$

Multi-language for FunTAL

$${}^{\tau}\mathcal{FT}(\mathbf{I}, \mathbf{H}) \mapsto^{*} {}^{\tau}\mathcal{FT}(\mathbf{v}) \mapsto \mathsf{v}$$

$$(\mathbf{I}, \mathbf{H}) : \mathbf{?}$$

$$(\mathbf{I}, \mathbf{H}) \mapsto^{*} \mathbf{?}$$

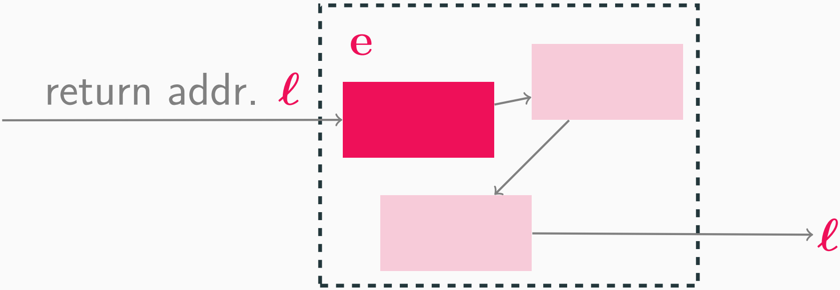
Multi-language for FunTAL

$$ {}^{\tau}\mathcal{FT}(\mathbf{I}, \mathbf{H}) \mapsto^{*} {}^{\tau}\mathcal{FT}(\mathbf{v}) \mapsto \mathsf{v} $$

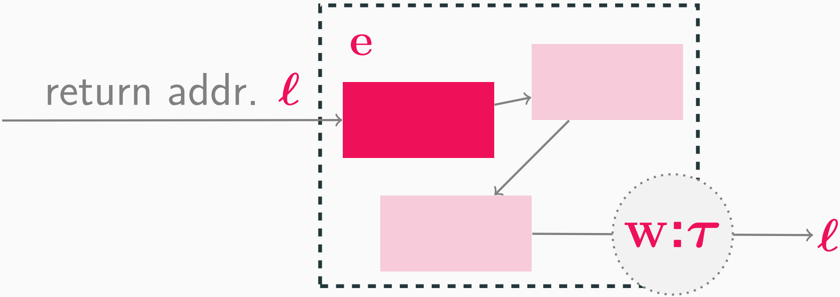
$$ (\mathbf{I}, \mathbf{H}) : \mathbf{?} $$

$$ (\mathbf{I}, \mathbf{H}) \mapsto^{*} \mathbf{?} $$

TAL components return to address



Return value passed to address



When will component $\mathbf{e}$ return $\boldsymbol{\tau}$?

When will component $\mathbf{e}$ return $\boldsymbol{\tau}$?

$$\boldsymbol{\Psi}; \boldsymbol{\Delta}; \boldsymbol{\chi}; \boldsymbol{\sigma}; \underset{\text{return marker}}{\mathbf{q}} \;\vdash (\mathbf{I}, \mathbf{H}) : \boldsymbol{\tau}, \boldsymbol{\sigma}'$$

When will component $\mathbf{e}$ return $\boldsymbol{\tau}$?

$$\boldsymbol{\Psi}; \boldsymbol{\Delta}; \chi; \boldsymbol{\sigma}; \mathtt{ra} \vdash (\mathbf{I}, \mathbf{H}) : \boldsymbol{\tau}, \boldsymbol{\sigma}'$$

$\{\mathtt{ra}:$ type of codeblock expecting to be passed a $\boldsymbol{\tau}$ and stack $\boldsymbol{\sigma}'\}$

When will component $\mathbf{e}$ return $\boldsymbol{\tau}$?

$$\boldsymbol{\Psi}; \boldsymbol{\Delta}; \chi; \boldsymbol{\sigma}; \mathtt{ra} \vdash \mathbf{(I, H)} : \boldsymbol{\tau}, \boldsymbol{\sigma'}$$

$\{\mathtt{ra}:$ type of codeblock expecting to be passed a $\boldsymbol{\tau}$ and stack $\boldsymbol{\sigma'}\}$

When will component $\mathbf{e}$ return $\boldsymbol{\tau}$?

$$\boldsymbol{\Psi}; \boldsymbol{\Delta}; \boldsymbol{\chi}; \boldsymbol{\sigma}; \mathbf{2} \;\vdash (\mathbf{I}, \mathbf{H}) : \boldsymbol{\tau}, \boldsymbol{\sigma}'$$

at $\mathbf{2}$: type of codeblock expecting to be passed a $\boldsymbol{\tau}$ and stack $\boldsymbol{\sigma}'$

What corresponds to Fun values?

$$\boldsymbol{\tau}\mathcal{FT}(\mathbf{I},\mathbf{H}) \mapsto^* \boldsymbol{\tau}\mathcal{FT}(\mathbf{v}) \mapsto \mathsf{\mathbf{v}}$$

What corresponds to Fun values?

$$\boldsymbol{\tau}\mathcal{FT}(\mathbf{I}, \mathbf{H}) \mapsto^* \ \boldsymbol{\tau}\mathcal{FT}(\mathbf{v}) \mapsto \mathsf{\mathbf{v}}$$

$$\mathbf{v} ::= (\mathtt{halt}\ \mathtt{r_d}, \mathbf{H})$$

What corresponds to Fun values?

$$^{\boldsymbol{\tau}}\mathcal{FT}(\mathbf{I}, \mathbf{H}) \mapsto^{*} {}^{\boldsymbol{\tau}}\mathcal{FT}(\mathbf{v}) \mapsto \mathsf{v}$$

$$\mathbf{v} ::= (\mathtt{halt}\ \mathtt{r_d}, \mathbf{H})$$

$$\boldsymbol{\Psi}; \boldsymbol{\Delta}; \boldsymbol{\chi}[\mathtt{r1} : \boldsymbol{\tau}]; \boldsymbol{\sigma}; \mathbf{end}\{\boldsymbol{\tau}; \boldsymbol{\sigma}\} \vdash (\mathtt{halt}\ \mathtt{r1}, \mathbf{H}) : \boldsymbol{\tau}, \boldsymbol{\sigma}$$

What corresponds to Fun values?

$$\boldsymbol{\tau}\mathcal{FT}(\mathbf{I}, \mathbf{H}) \mapsto^* \boldsymbol{\tau}\mathcal{FT}(\mathbf{v}) \mapsto \mathsf{v}$$

$$\mathbf{v} ::= (\texttt{halt}\ \texttt{r}_\texttt{d}, \mathbf{H})$$

$$\boldsymbol{\Psi}; \boldsymbol{\Delta}; \chi[\texttt{r1} : \boldsymbol{\tau}]; \boldsymbol{\sigma}; \mathbf{end}\{\boldsymbol{\tau}; \boldsymbol{\sigma}\} \vdash (\texttt{halt}\ \texttt{r1}, \mathbf{H}) : \boldsymbol{\tau}, \boldsymbol{\sigma}$$

What corresponds to Fun values?

$$^{\tau}\mathcal{FT}(\mathbf{I}, \mathbf{H}) \mapsto^* {}^{\tau}\mathcal{FT}(\mathbf{v}) \mapsto \mathbf{v}$$

$$\mathbf{v} ::= (\mathtt{halt}\ \mathtt{r_d}, \mathbf{H})$$

$$\boldsymbol{\Psi}; \boldsymbol{\Delta}; \chi[\mathtt{r1} : \boldsymbol{\tau}]; \boldsymbol{\sigma}; \mathbf{end}\{\boldsymbol{\tau}; \boldsymbol{\sigma}\} \vdash$$

$$(\mathtt{halt}\ \mathtt{r1}, \mathbf{H}) : \boldsymbol{\tau}, \boldsymbol{\sigma}$$

What corresponds to Fun values?

$$^{\tau}\mathcal{FT}(\mathbf{I}, \mathbf{H}) \mapsto^* \ ^{\tau}\mathcal{FT}(\mathbf{v}) \mapsto \mathbf{v}$$

$$\mathbf{v} ::= (\mathtt{halt}\ \mathtt{r}_{\mathtt{d}}, \mathbf{H})$$

$$\begin{gathered}\boldsymbol{\Psi}; \boldsymbol{\Delta}; \chi[\mathtt{r1} : \boldsymbol{\tau}]; \boldsymbol{\sigma}; \mathbf{end}\{\boldsymbol{\tau}; \boldsymbol{\sigma}\} \vdash \\ (\mathtt{halt}\ \mathtt{r1}, \mathbf{H}) : \boldsymbol{\tau}, \boldsymbol{\sigma}\end{gathered}$$

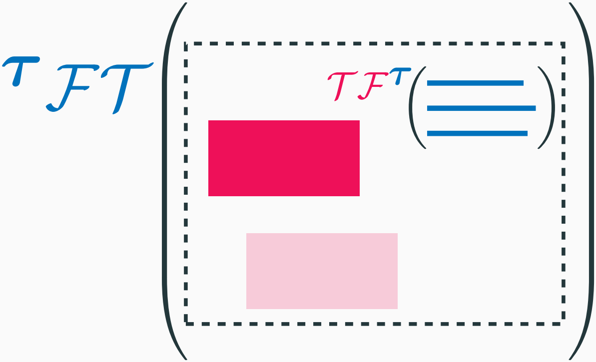
What corresponds to Fun values?

$$^{\tau}\mathcal{FT}(\mathbf{I}, \mathbf{H}) \mapsto^{*} {}^{\tau}\mathcal{FT}(\mathbf{v}) \mapsto \mathbf{v}$$

$$\mathbf{v} ::= (\mathtt{halt}\ \mathtt{r_d}, \mathbf{H})$$

$$\boldsymbol{\Psi}; \boldsymbol{\Delta}; \boldsymbol{\chi}[\mathtt{r1} : \boldsymbol{\tau}]; \boldsymbol{\sigma}; \mathbf{end}\{\boldsymbol{\tau}; \boldsymbol{\sigma}\} \vdash (\mathtt{halt}\ \mathtt{r1}, \mathbf{H}) : \boldsymbol{\tau}, \boldsymbol{\sigma}$$

Embedding Fun in TAL



How to embed an expression in TAL?

$$\texttt{import}\ \texttt{r}_{\texttt{d}}, \mathcal{TF}^{\tau}(\mathbf{v}) \mapsto \texttt{mv}\ \texttt{r}_{\texttt{d}}, \mathbf{w}$$

where $\mathbf{v}{:}\tau \rightsquigarrow \mathbf{w}{:}\tau^{+}$

How to embed an expression in TAL?

$$\mathtt{import}\ \mathtt{r_d}, \mathcal{TF}^{\tau}(\boxed{\mathbf{v}}) \mapsto \mathtt{mv}\ \mathtt{r_d}, \mathbf{w}$$

$$\text{where } \boxed{\mathbf{v}:\tau} \rightsquigarrow \mathbf{w}:\tau^{+}$$

How to embed an expression in TAL?

$$\texttt{import}\ \texttt{r}_\texttt{d}, \mathcal{TF}^{\boldsymbol{\tau}}(\mathbf{v}) \mapsto \texttt{mv}\ \texttt{r}_\texttt{d}, \mathbf{w}$$

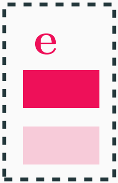
where $\mathbf{v}{:}\boldsymbol{\tau} \rightsquigarrow \mathbf{w}{:}\boldsymbol{\tau}^{+}$

How to embed an expression in TAL?

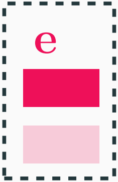
$$\mathtt{import}\ \mathtt{r_d}, \mathcal{TF}^{\tau}(\mathbf{v}) \mapsto \mathtt{mv}\ \mathtt{r_d}, \mathbf{w}$$

where $\mathbf{v}{:}\tau \rightsquigarrow \mathbf{w}{:}\tau^{+}$

So far, $\mathbf{q}$ can be $\mathtt{ra}$, $\mathbf{n}$, $\mathbf{end}\{\tau; \sigma\}$



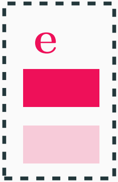
So far, $\mathbf{q}$ can be $\mathtt{ra}$, $\mathbf{n}$, $\mathbf{end}\{\boldsymbol{\tau};\boldsymbol{\sigma}\}$



$$\boldsymbol{\Psi}; \boldsymbol{\Delta}; \boldsymbol{\chi}; \boldsymbol{\sigma}; \mathtt{ra} \vdash \mathbf{e} : \boldsymbol{\tau}, \boldsymbol{\sigma}'$$

With these return markers, preconditions on $\mathbf{e}$ must specify all subsequent return markers.

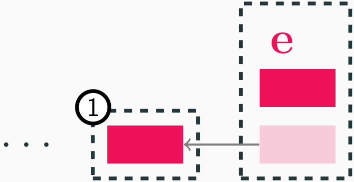
So far, $\mathbf{q}$ can be $\mathtt{ra}$, $\mathbf{n}$, $\mathbf{end}\{\boldsymbol{\tau};\boldsymbol{\sigma}\}$



$$\boldsymbol{\Psi};\boldsymbol{\Delta};\boldsymbol{\chi};\boldsymbol{\sigma};\mathtt{ra} \vdash \mathbf{e} : \boldsymbol{\tau},\boldsymbol{\sigma}'$$

With these return markers, preconditions on $\mathbf{e}$ must specify all subsequent return markers.

So far, $\mathbf{q}$ can be $\mathtt{ra}$, $\mathbf{n}$, $\mathbf{end}\{\tau;\sigma\}$



. . .

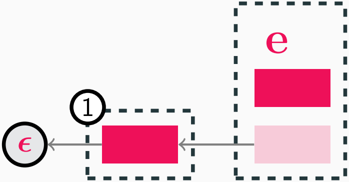
1

$$\underbrace{\mathtt{ra} : \forall[\Delta'].\{\chi';\sigma'\}^{\mathrm{q}}}$$

$$\Psi;\Delta;\chi;\sigma;\mathtt{ra} \vdash \mathbf{e} : \tau, \sigma'$$

With these return markers, preconditions on $\mathbf{e}$ must specify all subsequent return markers.

Components need polymorphic $\mathbf{q}$

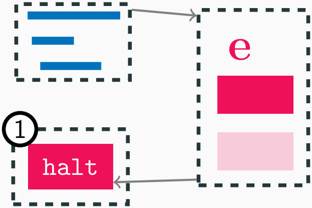


$$\underbrace{\mathtt{ra} : \forall[\Delta'].\{\chi'; \sigma'\}}$$

$$\Psi; \Delta, \epsilon; \chi; \sigma; \mathtt{ra} \vdash \mathbf{e} : \tau, \sigma'$$

With polymorphic return marker ϵ, caller instantiates with where control flows next.

Components need polymorphic $\mathbf{q}$



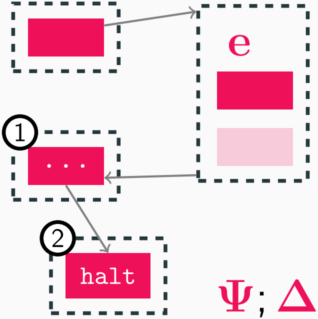
$$\boxed{\mathbf{end}\{\tau;\sigma'\}}$$

$$\underbrace{\mathtt{ra} : \forall[\Delta'].\{\chi';\sigma'\}^{\epsilon}}$$

$$\Psi; \Delta, \epsilon; \chi; \sigma; \mathtt{ra} \vdash \mathbf{e} : \tau, \sigma'$$

With polymorphic return marker ϵ, caller instantiates with where control flows next.

Components need polymorphic $\mathbf{q}$



$$\overbrace{\texttt{ra} : \forall[\Delta^{\dagger}].\{\chi^{\dagger}; \sigma^{\dagger}\}^{\mathbf{end}\{\tau;\sigma'\}}}^{②}$$

$$\overbrace{\texttt{ra} : \forall[\Delta'].\{\chi'; \sigma'\}}^{①} \quad \epsilon \quad \boxed{\texttt{ra}}$$

$$\Psi; \Delta, \epsilon; \chi; \sigma; \texttt{ra} \vdash \mathbf{e} : \tau, \sigma'$$

With polymorphic return marker ϵ, caller instantiates with where control flows next.

Questions we want to answer

How to safely mix assembly with high-level code?

How to reason about equivalence of mixed programs?

Questions we want to answer

How to safely mix assembly with high-level code? ✓

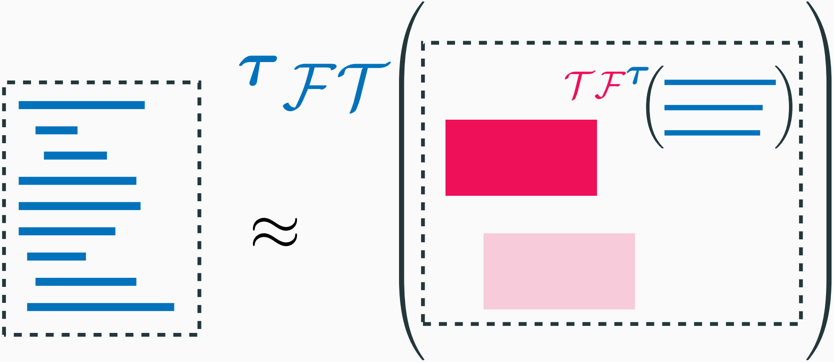
How to reason about equivalence of mixed programs?

Questions we want to answer

How to safely mix assembly with high-level code? ✓

How to reason about equivalence of mixed programs?

Proving program equivalence



Need logical relation for multi-language.

How logical relations work

$$\mathbf{e}{:}\boldsymbol{\tau} \approx {}^{\tau}\mathcal{F}\mathcal{T}(\mathrm{e}{:}\boldsymbol{\tau}^{+}) \;\; \text{means}$$

$$\mathbf{e} \mapsto^{*} \mathbf{v}_1 \iff \mathcal{F}\mathcal{T}(\mathrm{e}) \mapsto^{*} \mathbf{v}_2$$

$$\text{and } \mathbf{v}_1 \approx \mathbf{v}_2$$

How logical relations work

$$\mathbf{e{:}\tau} \approx^{\tau} \mathcal{FT}(\mathrm{e}{:}\boldsymbol{\tau}^{+}) \quad \text{means}$$

$$\mathbf{e} \mapsto^{*} \mathbf{v_1} \iff \mathcal{FT}(\mathrm{e}) \mapsto^{*} \mathbf{v_2}$$

and $\mathbf{v_1} \approx \mathbf{v_2}$

How logical relations work

$$\mathbf{e} : \boldsymbol{\tau} \approx^{\tau} \mathcal{FT}(\mathbf{e} : \boldsymbol{\tau}^{+}) \quad \text{means}$$

$$\mathbf{e} \mapsto^{*} \mathbf{v_1} \iff \mathcal{FT}(\mathrm{e}) \mapsto^{*} \mathbf{v_2}$$

and $\mathbf{v_1} \approx \mathbf{v_2}$

How logical relations work

$$\mathbf{e}{:}\boldsymbol{\tau} \approx^{\tau} \mathcal{FT}(\mathbf{e}{:}\boldsymbol{\tau}^{+}) \quad \text{means}$$

$$\mathbf{e} \mapsto^{*} \mathbf{v}_1 \iff \mathcal{FT}(\mathbf{e}) \mapsto^{*} \mathbf{v}_2$$

$$\text{and } \mathbf{v}_1 \approx \mathbf{v}_2$$

How logical relations work

$$\mathbf{e{:}\tau} \approx {}^{\tau}\mathcal{FT}(\mathbf{e{:}\tau^{+}}) \text{ means}$$

$$\mathbf{e} \mapsto^{*} \mathbf{v_1} \iff \mathcal{FT}(\mathbf{e}) \mapsto^{*} \mathbf{v_2}$$

and $\mathbf{v_1} \approx \mathbf{v_2}$

Write $\mathbf{v_1} \approx \mathbf{v_2}$ as $(\mathbf{v_1}, \mathbf{v_2}) \in \mathcal{V}(\boldsymbol{\tau})$.

Equivalence of functions

$$(\boldsymbol{\lambda}\mathbf{x}.\ \mathbf{e_1}, \boldsymbol{\lambda}\mathbf{x}.\ \mathbf{e_2}) \in \mathcal{V}(\boldsymbol{\tau_1} \rightarrow \boldsymbol{\tau_2})$$

Equivalence of functions

$$(\boldsymbol{\lambda} \mathbf{x}.\ \mathbf{e_1}, \boldsymbol{\lambda} \mathbf{x}.\ \mathbf{e_2}) \in \mathcal{V}(\boldsymbol{\tau_1} \to \boldsymbol{\tau_2})$$

"Related inputs result in related outputs"

Equivalence of functions

$$(\lambda \mathbf{x}.\ \mathbf{e}_1, \lambda \mathbf{x}.\ \mathbf{e}_2) \in \mathcal{V}(\tau_1 \to \tau_2)$$

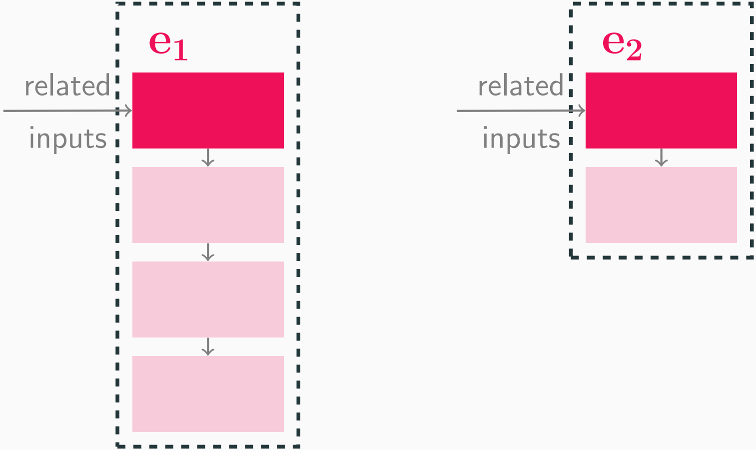
$$\text{if } (\mathbf{v}_1, \mathbf{v}_2) \in \mathcal{V}(\tau_1) \implies$$

$$\mathbf{e}_1[\mathbf{x} \mapsto \mathbf{v}_1] \approx \mathbf{e}_2[\mathbf{x} \mapsto \mathbf{v}_2]$$

"Related inputs result in related outputs"

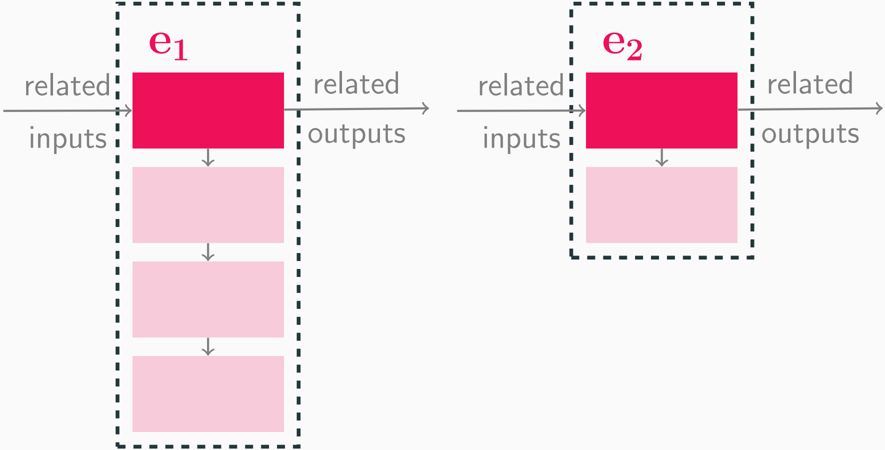
Equivalence of code blocks

$$\mathbf{e_1} \approx \mathbf{e_2}$$



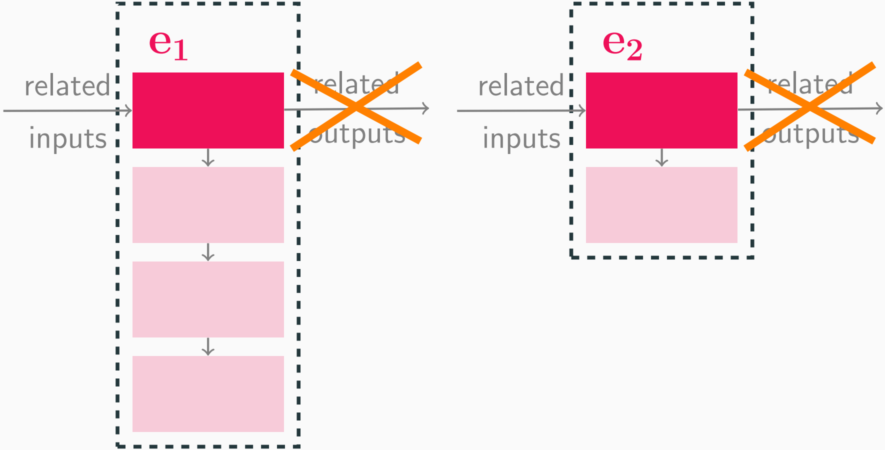
Equivalence of code blocks

$$\mathbf{e}_1 \approx \mathbf{e}_2$$



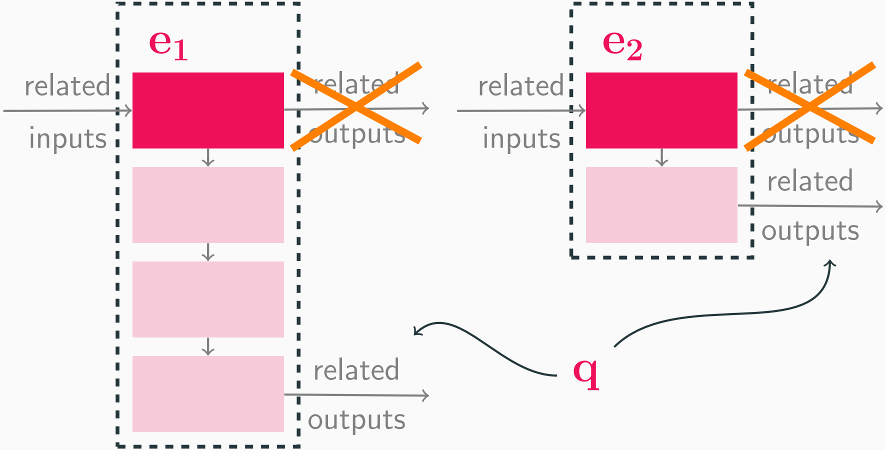
Equivalence of code blocks

$$\mathbf{e_1} \approx \mathbf{e_2}$$



Equivalence of code blocks

$$e_1 \approx e_2$$



Questions we want to answer

How to safely mix assembly with high-level code? ✓

How to reason about equivalence of mixed programs?

Questions we want to answer

How to safely mix assembly with high-level code? ✓

How to reason about equivalence of mixed programs? ✓

Future work

Future work

- Verification of some types of JIT transformations.

Future work

- Verification of some types of JIT transformations.
- Correctness for compilers targeting TAL (as suggested by *[Perconti-Ahmed '14]*).

Future work

- Verification of some types of JIT transformations.
- Correctness for compilers targeting TAL (as suggested by *[Perconti-Ahmed '14]*).
- Using return markers for slightly higher level (i.e., SSA-like) languages.

Conclusion

Return markers allow **safe mixing of components** where high-level code remains high-level and low-level remains low-level.

See paper for (much) more detail and a web-based interpreter for FunTAL.