A Kripke Logical Relation for Linear Functions:

The Story of a Free Theorem in the Presence of Non-termination

Phillip Mates & Amal Ahmed
Northeastern University

Programmers shouldn't have to think about compilers

Fully abstract compiler

Source $\qquad\qquad$ Target

$$\mathsf{e} : \tau \quad \rightsquigarrow \quad \mathbf{e} : \tau^{+}$$

Fully abstract compiler

Source **Target**

$$\mathsf{e} : \tau \quad \rightsquigarrow \quad \mathbf{e} : \tau^{+}$$

preserves equivalence

$$\mathsf{e}_1 \approx_S^{ctx} \mathsf{e}_2 : \tau \implies \mathbf{e}_1 \approx_T^{ctx} \mathbf{e}_2 : \tau^{+}$$

Fully abstract compiler

Source **Target**

$$\mathsf{e} : \tau \quad \leadsto \quad \mathbf{e} : \tau^{+}$$

preserves equivalence

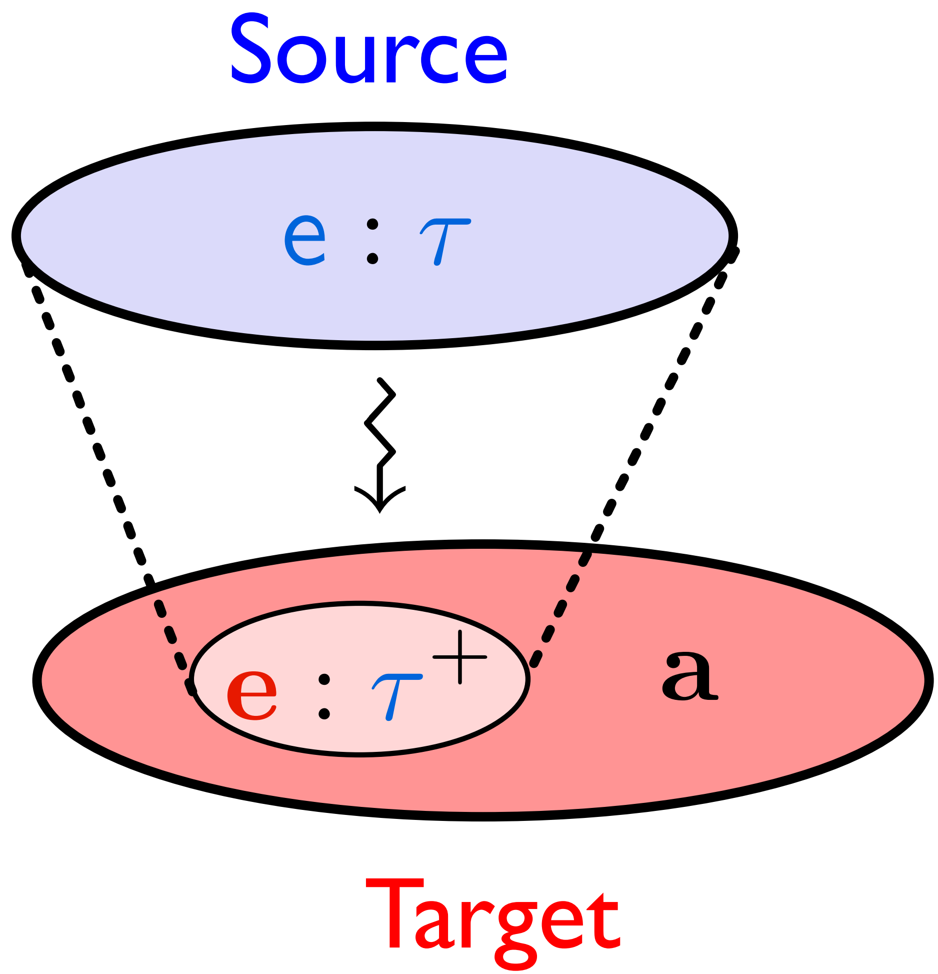
$$\mathsf{e}_1 \approx_S^{ctx} \mathsf{e}_2 : \tau \implies \mathbf{e}_1 \approx_T^{ctx} \mathbf{e}_2 : \tau^{+}$$

&

reflects equivalence

$$\mathsf{e}_1 \approx_S^{ctx} \mathsf{e}_2 : \tau \impliedby \mathbf{e}_1 \approx_T^{ctx} \mathbf{e}_2 : \tau^{+}$$

Verifying a CPS translation fully abstract



Outline

Finding a fully abstract CPS translation w/ $\mu\,\alpha.\tau$

- Standard CPS ✗
- Polymorphic CPS (✓ only in a pure setting)
- Linear + polymorphic CPS

Proving full-abstraction

Towards a semantic model for linearly-treated functions

Standard CPS isn't fully abstract

$$\mathsf{e} : \tau \;\rightsquigarrow\; \mathbf{e} : (\tau^{+} \to \mathrm{ans}) \to \mathrm{ans}$$

$$\begin{aligned} \mathsf{int}^{+} &= \mathbf{int} \\ (\tau_1 \to \tau_2)^{+} &= \tau_1{}^{+} \to ((\tau_2{}^{+} \to \mathrm{ans}) \to \mathrm{ans}) \end{aligned}$$

Standard CPS isn't fully abstract

$$\mathsf{e} : \tau \;\; \rightsquigarrow \;\; \mathbf{e} : (\tau^{+} \to \mathrm{ans}) \to \mathrm{ans}$$

$$\mathsf{v}_1 = \lambda(\mathsf{f}, \mathsf{g})\,.\,\mathsf{f}\ 0; \mathsf{g}\ 0; 0$$
$$\mathsf{v}_2 = \lambda(\mathsf{f}, \mathsf{g})\,.\,\mathsf{g}\ 0; \mathsf{f}\ 0; 0$$

Standard CPS isn't fully abstract

$$\mathsf{e} : \tau \;\rightsquigarrow\; \mathbf{e} : (\tau^{+} \to \mathrm{ans}) \to \mathrm{ans}$$

$\mathsf{f}, \mathsf{g} : \mathsf{int} \to \mathsf{int}$

$$\mathsf{v}_1 = \lambda(\mathsf{f}, \mathsf{g})\,.\,\mathsf{f}\ 0; \mathsf{g}\ 0; 0$$
$$\mathsf{v}_2 = \lambda(\mathsf{f}, \mathsf{g})\,.\,\mathsf{g}\ 0; \mathsf{f}\ 0; 0$$

Standard CPS isn't fully abstract

$$\mathsf{e} : \tau \;\rightsquigarrow\; \mathbf{e} : (\tau^{+} \to \mathrm{ans}) \to \mathrm{ans}$$

$\mathsf{f}, \mathsf{g} : \mathsf{int} \to \mathsf{int}$

$$\mathsf{v}_1 = \lambda(\mathsf{f}, \mathsf{g})\,.\,\mathsf{f}\ 0; \mathsf{g}\ 0; 0$$
$$\mathsf{v}_2 = \lambda(\mathsf{f}, \mathsf{g})\,.\,\mathsf{g}\ 0; \mathsf{f}\ 0; 0$$

$$\mathbf{v}_1 = \lambda(\mathbf{f}, \mathbf{g}, \mathbf{k} : \mathsf{int} \to \mathrm{ans})\,.\,(\mathbf{f}\ \mathbf{0})(\lambda\_.(\mathbf{g}\ \mathbf{0})(\lambda\_.\mathbf{k}\ \mathbf{0}))$$
$$\mathbf{v}_2 = \lambda(\mathbf{f}, \mathbf{g}, \mathbf{k} : \mathsf{int} \to \mathrm{ans})\,.\,(\mathbf{g}\ \mathbf{0})(\lambda\_.(\mathbf{f}\ \mathbf{0})(\lambda\_.\mathbf{k}\ \mathbf{0}))$$

Standard CPS isn't fully abstract

$$\mathsf{e} : \tau \quad \rightsquigarrow \quad \mathbf{e} : (\tau^{+} \to \mathrm{ans}) \to \mathrm{ans}$$

$$\mathsf{f}, \mathsf{g} : \mathsf{int} \to \mathsf{int}$$

$$\mathsf{v}_1 = \lambda(\mathsf{f}, \mathsf{g})\,.\,\mathsf{f}\ 0; \mathsf{g}\ 0; 0$$

$$\mathsf{v}_2 = \lambda(\mathsf{f}, \mathsf{g})\,.\,\mathsf{g}\ 0; \mathsf{f}\ 0; 0$$

$$\mathbf{f}, \mathbf{g} : \mathsf{int} \to ((\mathsf{int} \to \mathrm{ans}) \to \mathrm{ans})$$

$$\mathbf{v}_1 = \lambda(\mathbf{f}, \mathbf{g}, \mathbf{k} : \mathsf{int} \to \mathrm{ans})\,.\,(\mathbf{f}\ \mathbf{0})(\lambda\_.(\mathbf{g}\ \mathbf{0})(\lambda\_.\mathbf{k}\ \mathbf{0}))$$

$$\mathbf{v}_2 = \lambda(\mathbf{f}, \mathbf{g}, \mathbf{k} : \mathsf{int} \to \mathrm{ans})\,.\,(\mathbf{g}\ \mathbf{0})(\lambda\_.(\mathbf{f}\ \mathbf{0})(\lambda\_.\mathbf{k}\ \mathbf{0}))$$

Standard CPS isn't fully abstract

$$\mathsf{e} : \tau \;\rightsquigarrow\; \mathbf{e} : (\tau^{+} \to \mathrm{ans}) \to \mathrm{ans}$$

$$\mathbf{C} = \lambda \mathbf{k}.[\cdot](\lambda\_.\lambda\_.(\mathbf{k}\ \mathbf{1}), \lambda\_.\lambda\_.(\mathbf{k}\ \mathbf{2}), \mathbf{k})$$

$$\mathbf{v}_1 = \lambda(\mathbf{f}, \mathbf{g}, \mathbf{k} : \mathsf{int} \to \mathrm{ans}) \,.\, (\mathbf{f}\ \mathbf{0})(\lambda\_.(\mathbf{g}\ \mathbf{0})(\lambda\_.\mathbf{k}\ \mathbf{0}))$$

$$\mathbf{v}_2 = \lambda(\mathbf{f}, \mathbf{g}, \mathbf{k} : \mathsf{int} \to \mathrm{ans}) \,.\, (\mathbf{g}\ \mathbf{0})(\lambda\_.(\mathbf{f}\ \mathbf{0})(\lambda\_.\mathbf{k}\ \mathbf{0}))$$

Standard CPS isn't fully abstract

$$\mathsf{e} : \tau \rightsquigarrow \mathbf{e} : (\tau^{+} \to \mathrm{ans}) \to \mathrm{ans}$$

$$\mathbf{C} = \lambda\mathbf{k}.[\cdot](\lambda\_.\lambda\_.(\mathbf{k}\ \mathbf{1}), \lambda\_.\lambda\_.(\mathbf{k}\ \mathbf{2}), \mathbf{k})$$

$$\mathbf{C}\,[\mathbf{v_1}]\,id \Downarrow \mathbf{1}$$

$$\mathbf{v}_1 = \lambda(\mathbf{f}, \mathbf{g}, \mathbf{k} : \mathsf{int} \to \mathrm{ans})\,.\,(\mathbf{f}\ \mathbf{0})(\lambda\_.(\mathbf{g}\ \mathbf{0})(\lambda\_.\mathbf{k}\ \mathbf{0}))$$

$$\mathbf{v}_2 = \lambda(\mathbf{f}, \mathbf{g}, \mathbf{k} : \mathsf{int} \to \mathrm{ans})\,.\,(\mathbf{g}\ \mathbf{0})(\lambda\_.(\mathbf{f}\ \mathbf{0})(\lambda\_.\mathbf{k}\ \mathbf{0}))$$

Standard CPS isn't fully abstract

$$\mathsf{e} : \tau \quad \rightsquigarrow \quad \mathbf{e} : (\tau^{+} \to \mathrm{ans}) \to \mathrm{ans}$$

$$\mathbf{C} = \lambda \mathbf{k}.[\cdot](\lambda\_.\lambda\_.(\mathbf{k}\ \mathbf{1}), \lambda\_.\lambda\_.(\mathbf{k}\ \mathbf{2}), \mathbf{k})$$

$$\mathbf{C}\,[\mathbf{v_1}]\,id \Downarrow \mathbf{1} \qquad \mathbf{C}\,[\mathbf{v_2}]\,id \Downarrow \mathbf{2}$$

$$\mathbf{v}_1 = \lambda(\mathbf{f}, \mathbf{g}, \mathbf{k} : \mathsf{int} \to \mathrm{ans})\,.\,(\mathbf{f}\ \mathbf{0})(\lambda\_.(\mathbf{g}\ \mathbf{0})(\lambda\_.\mathbf{k}\ \mathbf{0}))$$

$$\mathbf{v}_2 = \lambda(\mathbf{f}, \mathbf{g}, \mathbf{k} : \mathsf{int} \to \mathrm{ans})\,.\,(\mathbf{g}\ \mathbf{0})(\lambda\_.(\mathbf{f}\ \mathbf{0})(\lambda\_.\mathbf{k}\ \mathbf{0}))$$

How can we modify the standard CPS translation to be fully abstract?

Standard CPS

$$\mathsf{e} : \tau \;\rightsquigarrow\; \mathbf{e} : (\tau^{+} \to \mathrm{ans}) \to \mathrm{ans}$$

$$\mathsf{int}^{+} = \mathbf{int}$$

$$(\tau_1 \to \tau_2)^{+} = {\tau_1}^{+} \to ((\tau_2{}^{+} \to \mathrm{ans}) \to \mathrm{ans})$$

Ahmed & Blume's polymorphic CPS

$$\mathsf{e} : \tau \;\rightsquigarrow\; \mathbf{e} : \forall\alpha.(\tau^{+} \to \alpha) \to \alpha$$

$$\mathsf{int}^{+} = \mathbf{int}$$

$$(\tau_1 \to \tau_2)^{+} = {\tau_1}^{+} \to (\forall\alpha\,.\,(\tau_2{}^{+} \to \alpha) \to \alpha)$$

Standard CPS

$$\mathsf{e} : \tau \;\rightsquigarrow\; \mathbf{e} : (\tau^{+} \to \mathrm{ans}) \to \mathrm{ans}$$

$$\mathsf{int}^{+} = \mathbf{int}$$

$$(\tau_1 \to \tau_2)^{+} = {\tau_1}^{+} \to ((\tau_2{}^{+} \to \mathrm{ans}) \to \mathrm{ans})$$

Ahmed & Blume's polymorphic CPS

$$\mathsf{e} : \tau \;\rightsquigarrow\; \mathbf{e} : \forall \alpha . (\tau^{+} \to \alpha) \to \alpha$$

$$\mathsf{int}^{+} = \mathbf{int}$$

$$(\tau_1 \to \tau_2)^{+} = {\tau_1}^{+} \to (\forall \alpha . (\tau_2{}^{+} \to \alpha) \to \alpha)$$

What happens when we add non-termination?

Standard CPS

$$\mathsf{e} : \tau \;\rightsquigarrow\; \mathbf{e} : (\tau^{+} \to \mathrm{ans}) \to \mathrm{ans}$$

$$\mathsf{int}^{+} = \mathbf{int}$$

$$(\tau_1 \to \tau_2)^{+} = \tau_1{}^{+} \to ((\tau_2{}^{+} \to \mathrm{ans}) \to \mathrm{ans})$$

Ahmed & Blume's polymorphic CPS

$$\mathsf{e} : \tau \;\rightsquigarrow\; \mathbf{e} : \forall\alpha.(\tau^{+} \to \alpha) \to \alpha$$

$$\mathsf{int}^{+} = \mathbf{int}$$

$$(\tau_1 \to \tau_2)^{+} = \tau_1{}^{+} \to (\forall\alpha\,.\,(\tau_2{}^{+} \to \alpha) \to \alpha)$$

Linear + polymorphic CPS

$$\mathsf{e} : \tau \;\rightsquigarrow\; \mathbf{e} : \forall\alpha.(\tau^{+} \to \alpha) \multimap \alpha$$

$$\mathsf{int}^{+} = \mathbf{int}$$

$$(\tau_1 \to \tau_2)^{+} = \tau_1{}^{+} \to (\forall\alpha\,.\,(\tau_2{}^{+} \to \alpha) \multimap \alpha)$$

Outline

Finding a fully abstract CPS translation w/ $\mu\,\alpha.\tau$

Proving full-abstraction

- Polymorphic CPS proof
- Scaling to a non-terminating setting using linearity

Towards a semantic model for linearly-treated functions

Proving polymorphic CPS fully abstract

Ahmed + Blume's approach relies on
a **type isomorphism**:

$$\tau \cong \forall \alpha . (\tau \to \alpha) \to \alpha$$

Proving polymorphic CPS fully abstract

The type isomorphism relies on
a **parametric condition**:

$$\begin{aligned} f &: \forall\alpha.(\tau \to \alpha) \to \alpha \\ k &: \tau \to \tau_k \\ id &: \tau \to \tau \end{aligned}$$

$$f\ [\tau_k]\ k \quad \approx^{ctx} \quad k\ (f\ [\tau]\ id) : \tau_k$$

Parametric condition fails in the presence of non-termination

$$f\ [\tau_k]\ k \quad \overset{?}{\approx}^{ctx} \quad k\ (f\ [\tau]\ id) : \tau_k$$

Parametric condition fails in the presence of non-termination

$f : \forall\alpha.(\mathsf{int} \to \alpha) \to \alpha$

$f = \lambda[\alpha](x_k : \mathsf{int} \to \alpha).(x_k\ 0); (x_k\ 1)$

$k : \mathsf{int} \to \tau_k$

$k = \lambda(n : \mathsf{int}).\mathit{if0}\ n\ \mathit{then}\ \Omega\ \mathit{else}\ n$

$$f\ [\tau_k]\ k \quad \overset{?}{\approx}^{ctx} \quad k\ (f\ [\mathsf{int}]\ id) : \tau_k$$

Parametric condition fails in the presence of non-termination

$$f : \forall\alpha.(\mathsf{int} \to \alpha) \to \alpha$$

$$f = \lambda[\alpha](x_k : \mathsf{int} \to \alpha).(x_k\ 0); (x_k\ 1)$$

$$k : \mathsf{int} \to \tau_k$$

$$k = \lambda(n : \mathsf{int}).\mathit{if0}\ n\ \mathit{then}\ \Omega\ \mathit{else}\ n$$

$$f\ [\tau_k]\ k \quad \stackrel{?}{\approx}^{ctx} \quad k\ (f\ [\mathsf{int}]\ id) : \tau_k$$

$$\Uparrow$$

Parametric condition fails in the presence of non-termination

$$f : \forall\alpha.(\mathsf{int} \to \alpha) \to \alpha$$

$$f = \lambda[\alpha](x_k : \mathsf{int} \to \alpha).(x_k\ 0); (x_k\ 1)$$

$$k : \mathsf{int} \to \tau_k$$

$$k = \lambda(n : \mathsf{int}).\mathit{if0}\ n\ \mathit{then}\ \Omega\ \mathit{else}\ n$$

$$f\ [\tau_k]\ k \quad \not\approx^{ctx} \quad k\ (f\ [\mathsf{int}]\ id) : \tau_k$$

$$\Uparrow \qquad\qquad\qquad \Downarrow 1$$

A step back

Ahmed & Blume's polymorphic CPS

$$\mathrm{e} : \tau \quad \rightsquigarrow \quad \mathbf{e} : \forall \alpha. (\tau^{+} \to \alpha) \to \alpha$$

A step back

Ahmed & Blume's polymorphic CPS

$$\mathsf{e} : \tau \;\rightsquigarrow\; \mathbf{e} : \forall\alpha.(\tau^{+} \to \alpha) \to \alpha$$

requires

$$\tau \cong \forall\alpha.(\tau \to \alpha) \to \alpha$$

A step back

Ahmed & Blume's polymorphic CPS

$$\mathsf{e} : \tau \;\rightsquigarrow\; \mathbf{e} : \forall \alpha. (\tau^{+} \to \alpha) \to \alpha$$

requires

$$\tau \cong \forall \alpha. (\tau \to \alpha) \to \alpha$$

A step back

Ahmed & Blume's polymorphic CPS

$$\mathsf{e} : \tau \;\; \rightsquigarrow \;\; \mathbf{e} : \forall\alpha.(\tau^{+} \to \alpha) \to \alpha$$

requires

$$\tau \cong \forall\alpha.(\tau \to \alpha) \to \alpha$$

Linear + polymorphic CPS

$$\mathsf{e} : \tau \;\; \rightsquigarrow \;\; \mathbf{e} : \forall\alpha.(\tau^{+} \to \alpha) \multimap \alpha$$

A step back

Ahmed & Blume's polymorphic CPS

$$\mathsf{e} : \tau \;\rightsquigarrow\; \mathbf{e} : \forall\alpha.(\tau^{+} \to \alpha) \to \alpha$$

requires

$$\tau \cong \forall\alpha.(\tau \to \alpha) \to \alpha$$

Linear + polymorphic CPS

$$\mathsf{e} : \tau \;\rightsquigarrow\; \mathbf{e} : \forall\alpha.(\tau^{+} \to \alpha) \multimap \alpha$$

requires

$$\tau \cong \forall\alpha.(\tau \to \alpha) \multimap \alpha$$

A step back

Ahmed & Blume's polymorphic CPS

$$\mathsf{e} : \tau \quad \rightsquigarrow \quad \mathbf{e} : \forall\alpha.(\tau^{+} \to \alpha) \to \alpha$$

requires

$$\tau \cong \forall\alpha.(\tau \to \alpha) \to \alpha$$

Ω

Linear + polymorphic CPS

$$\mathsf{e} : \tau \quad \rightsquigarrow \quad \mathbf{e} : \forall\alpha.(\tau^{+} \to \alpha) \multimap \alpha$$

requires

linear

$$\tau \cong \forall\alpha.(\tau \to \alpha) \multimap \alpha$$

A "linear" parametric condition in presence of non-termination

linear

$$
\begin{aligned}
f &: \forall\alpha.(\tau \to \alpha) \multimap \alpha \\
k &: \tau \to \tau_k \\
id &: \tau \to \tau
\end{aligned}
$$

$$f\ [\tau_k]\ k \quad \approx^{ctx} \quad k\ (f\ [\tau]\ id) : \tau_k$$

How do you prove this
new parametric condition,
which uses continuations linearly?

Free theorem: use a logical relation

$$e_1 \approx^{log} e_2 : \tau \;\; =$$

$$e_1 \precsim^{log} e_2 : \tau \quad \wedge \quad e_2 \precsim^{log} e_1 : \tau$$

Free theorem: use a logical relation

$$e_1 \approx^{log} e_2 : \tau \;\; =$$

$$e_1 \precsim^{log} e_2 : \tau \quad \wedge \quad e_2 \precsim^{log} e_1 : \tau$$

$$e_1 \precsim^{log} e_2 : \tau \;\; =$$

$$\forall v_1 \;.\; e_1 \Downarrow v_1 \implies$$

$$\exists v_2 \;.\; e_2 \Downarrow v_2 \quad \wedge \quad v_1 \precsim^{log} v_2 : \tau$$

Free theorem: use a logical relation

$$\frac{f\ [\tau_k]\ k \quad \approx^{log} \quad k\ (f\ [\tau]\ id) : \tau_k}{\quad}$$

$$\exists v_2\ .\ e_2 \Downarrow v_2 \quad \wedge \quad v_1 \precsim^{log} v_2 : \tau$$

Free theorem: use a logical relation

log

$$\frac{f\ [\tau_k]\ k \quad \approx^{log} \quad k\ (f\ [\tau]\ id) : \tau_k}{f\ [\tau_k]\ k \quad \precsim^{log} \quad k\ (f\ [\tau]\ id) : \tau_k}$$

$$\exists v_2\ .\ e_2 \Downarrow v_2 \quad \wedge \quad v_1 \precsim^{log} v_2 : \tau$$

Free theorem: use a logical relation

$$\frac{f\ [\tau_k]\ k \quad \approx^{log} \quad k\ (f\ [\tau]\ id) : \tau_k}{\begin{array}{l} f\ [\tau_k]\ k \quad \precsim^{log} \quad k\ (f\ [\tau]\ id) : \tau_k \quad ✓ \\ k\ (f\ [\tau]\ id) \quad \precsim^{log} \quad f\ [\tau_k]\ k : \tau_k \quad ? \end{array}}$$

$$\exists v_2\ .\ e_2 \Downarrow v_2 \quad \wedge \quad v_1 \precsim^{log} v_2 : \tau$$

Proving $k\ (f\ [\tau]\ id) \quad \precsim^{log} \quad f\ [\tau_k]\ k : \tau_k$

Proving $k\ (f\ [\tau]\ id) \quad \precsim^{log} \quad f\ [\tau_k]\ k : \tau_k$

know $f\ [\tau]\ id \longmapsto^* v_{id}$

Proving $k\ (f\ [\tau]\ id) \quad \precsim^{log} \quad f\ [\tau_k]\ k : \tau_k$

know $f\ [\tau]\ id \longmapsto^* v_{id}$

$k\ v_{id} \longmapsto^* v_1$

Proving $k\ (f\ [\tau]\ id) \quad \precsim^{log} \quad f\ [\tau_k]\ k : \tau_k$

know $f\ [\tau]\ id \longmapsto^* v_{id}$

$k\ v_{id} \longmapsto^* v_1$

show $f\ [\tau_k]\ k \longmapsto^* v_2$

Proving $k\ (f\ [\tau]\ id) \precsim^{log} f\ [\tau_k]\ k : \tau_k$

know $f\ [\tau]\ id \longmapsto^* v_{id}$

$k\ v_{id} \longmapsto^* v_1$

show $f\ [\tau_k]\ k \longmapsto^* v_2$

$v_1 \precsim^{log} v_2$

Proving $k\ (f\ [\tau]\ id) \precsim^{log} f\ [\tau_k]\ k : \tau_k$

know $f\ [\tau]\ id \longmapsto^* v_{id}$

$k\ v_{id} \longmapsto^* v_1$

show $f\ [\tau_k]\ k \longmapsto^* v_2$

$v_1 \precsim^{log} v_2$

$\boxed{\text{know: } \quad f \precsim^{log} f : \forall\alpha.(\tau \to \alpha) \multimap \alpha}$

Proving $k\ (f\ [\tau]\ id) \precsim^{log} f\ [\tau_k]\ k : \tau_k$

know $f\ [\tau]\ id \longmapsto^* v_{id}$

$k\ v_{id} \longmapsto^* v_1$

show $f\ [\tau_k]\ k \longmapsto^* v_2$

$v_1 \precsim^{log} v_2$

know: $f \precsim^{log} f : \forall\alpha.(\tau \to \alpha) \multimap \alpha$

show: $id \precsim^{log} k : \tau \to \alpha$

Proving $k\ (f\ [\tau]\ id) \precsim^{log} f\ [\tau_k]\ k : \tau_k$

know $f\ [\tau]\ id \longmapsto^* v_{id}$

$k\ v_{id} \longmapsto^* v_1$

show $f\ [\tau_k]\ k \longmapsto^* v_2$

$v_1 \precsim^{log} v_2$

know: $f \precsim^{log} f : \forall\alpha.(\tau \to \alpha) \multimap \alpha$

show: $id \precsim^{log} k : \tau \to \alpha \mid \alpha \mapsto (\tau, \tau_k, R)$

Proving $id \ \precsim^{log} \ k : \tau \to \alpha$

Proving $id \precsim^{log} k : \tau \to \alpha$

know $v_1 \precsim^{log} v_2 : \tau$

Proving $id \ \precsim^{log} \ k : \tau \to \alpha$

know $v_1 \precsim^{log} v_2 : \tau$

know $id \; v_1 \Downarrow$

Proving $id \ \precsim^{log} \ k : \tau \to \alpha$

know $v_1 \precsim^{log} v_2 : \tau$

know $id \ v_1 \Downarrow$

show $k \ v_2 \Downarrow$

Proving $id \precsim^{log} k : \tau \to \alpha$

know $v_1 \precsim^{log} v_2 : \tau$

know $id\ v_1 \Downarrow$

show $k\ v_2 \Downarrow$

behaviour?

Proving $id \precsim^{log} k : \tau \to \alpha$

know $v_1 \precsim^{log} v_2 : \tau$

know $id\ v_1 \Downarrow$

show $k\ v_2 \Downarrow$

$$f : \forall \alpha. (\tau \to \alpha) \multimap \alpha$$

$$k\ (f\ [\tau]\ \color{red}{id}) \quad \precsim^{log} \quad f\ [\tau_k]\ \color{red}{k} : \tau_k$$

Proving $id \precsim^{log} k : \tau \to \alpha$

know $v_1 \precsim^{log} v_2 : \tau$

know $id\ v_1 \Downarrow$

show $k\ v_2 \Downarrow$

$$f : \forall \alpha.(\tau \to \alpha) \multimap \alpha$$

$$k\ (f\ [\tau]\ id) \quad \precsim^{log} \quad f\ [\tau_k]\ k : \tau_k$$

$$(f\ [\tau]\ id) \longmapsto^* v_{id}$$

Proving $id \precsim^{log} k : \tau \to \alpha$

know $v_1 \precsim^{log} v_2 : \tau$

know $id\ v_1 \Downarrow$ show $k\ v_2 \Downarrow$

$=$

$$f : \forall\alpha.(\tau \to \alpha) \multimap \alpha$$

$$k\ (f\ [\tau]\ id) \quad \precsim^{log} \quad f\ [\tau_k]\ k : \tau_k$$

$$(f\ [\tau]\ id) \longmapsto^* v_{id}$$

Outline

Finding a fully abstract CPS translation w/ $\mu\,\alpha.\tau$

Proving full-abstraction

Towards a semantic model for linearly-treated functions

Function semantics: case 1

$$f : (\tau_1 \to \tau_2) \to \tau_3 \qquad g,\, g' : \; \tau_1 \to \tau_2$$

$$f\ g \precsim^{log} f\ g' : \tau_3$$

$$\boxed{g \precsim^{log} g' : \tau_1 \to \tau_2 \ \textbf{?}}$$

Function semantics: case 1

$$f : (\tau_1 \rightarrow \tau_2) \rightarrow \tau_3 \qquad g,\, g' : \tau_1 \rightarrow \tau_2$$

$$f\ g \precsim^{log} f\ g' : \tau_3$$

$$g \precsim^{log} g' : \underbrace{\tau_1 \rightarrow \tau_2}_{\text{unrestricted usage}}$$

for all possible inputs

Function semantics: case 2

$$f : (\tau_1 \multimap \tau_2) \rightarrow \tau_3 \qquad g,\ g' : \ \tau_1 \multimap \tau_2$$

$$f\ g \precsim^{log} f\ g' : \tau_3$$

$$g \precsim^{log} g' : \underbrace{\tau_1 \multimap \tau_2}_{\text{unrestricted usage}}$$

for all possible inputs & treat inputs linearly

Function semantics: case 3

$$f : (\tau_1 \to \tau_2) \multimap \tau_3 \qquad g,\ g' : \ \tau_1 \to \tau_2$$

$$f\ g \precsim^{log} f\ g' : \tau_3$$

$$g \precsim^{log} g' : \underbrace{\tau_1 \to \tau_2}_{\text{linearly used}}$$

for particular inputs

Function semantics: case 4

$$f : (\tau_1 \multimap \tau_2) \multimap \tau_3 \qquad g,\ g' : \ \tau_1 \multimap \tau_2$$

$$f\ g \precsim^{log} f\ g' : \tau_3$$

$$g \precsim^{log} g' : \underbrace{\tau_1 \multimap \tau_2}_{\text{linearly used}}$$

for particular inputs & treat input linearly

Relating linearly-treated functions

$$id \precsim^{log} k : \tau \to \alpha$$

Relating linearly-treated functions

$$id \precsim^{log} k : \tau \to \alpha$$

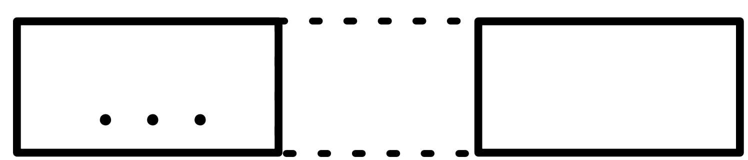
$$f \precsim^{log} f : \forall \alpha.(\tau \to \alpha) \multimap \alpha$$

$$W.\mathcal{I} \;=\; \boxed{\;\ldots\;}$$

Relating linearly-treated functions

$$id \precsim^{log} k : \tau \to \alpha$$

$$f \precsim^{log} f : \forall \alpha.(\tau \to \alpha) \multimap \alpha$$

$W'.\mathcal{I} \; =$ 

Relating linearly-treated functions

$$id \precsim^{log} k : \tau \to \alpha$$

$$f \precsim^{log} f : \forall \alpha.(\tau \to \alpha) \multimap \alpha$$

$$W'.\mathcal{I} = \boxed{\quad \ldots \quad} \cdots \boxed{\quad P \quad}$$

Relating linearly-treated functions

$$\boxed{id \precsim^{log} k} : \tau \to \alpha$$

$$v_1 \precsim^{log} v_2 : \tau \quad \text{where} \quad P(v_1, v_2)$$

$$P(v_1, v_2) \quad \text{iff} \quad v_1 = v_{id}$$

$$W'.\mathcal{I} \;=\; \boxed{\;\ldots\;} \cdots \boxed{\;P\;}$$

Linearity by default

$$\tau \rightarrow \tau' \;=\; !\tau \multimap \tau'$$

Kripke logical relation for linearity

$$f_1 \precsim_W^{log} f_2 : \tau \multimap \tau' \ =$$

$$W.\mathcal{I} \ = \ \boxed{}$$

Kripke logical relation for linearity

$$f_1 \precsim_W^{log} f_2 : \tau \multimap \tau' \;\; =$$

$$W.\mathcal{I} \;\; = \;\; \boxed{\;P_f\;}$$

Kripke logical relation for linearity

$$f_1 \precsim_W^{log} f_2 : \tau \multimap \tau' \ =$$

$$\forall v_1, v_2 \ . \ v_1 \precsim_{W'}^{log} v_2 : \tau \ \wedge \ P_f(v_1, v_2) \implies$$

$$f_1\, v_1 \precsim_{W'}^{log} f_2\, v_2 : \tau'$$

$$W.\mathcal{I} \ = \ \boxed{P_f}$$

Kripke logical relation for linearity

$$f_1 \precsim^{log}_{W} f_2 : \tau \multimap \tau' \;=$$

$$\forall v_1, v_2 \;.\; v_1 \precsim^{log}_{W'} v_2 : \tau \;\wedge\; P_f(v_1, v_2) \implies$$

$$f_1\, v_1 \precsim^{log}_{W'} f_2\, v_2 : \tau'$$

$$W'.\mathcal{I} \;=\; \boxed{\;P_f\;} \cdots \boxed{}$$

Kripke logical relation for linearity

$$f_1 \precsim^{log}_{W} f_2 : \tau \multimap \tau' \;=$$

$$\forall v_1, v_2 \,.\; v_1 \precsim^{log}_{W'} v_2 : \tau \;\wedge\; P_f(v_1, v_2) \implies$$

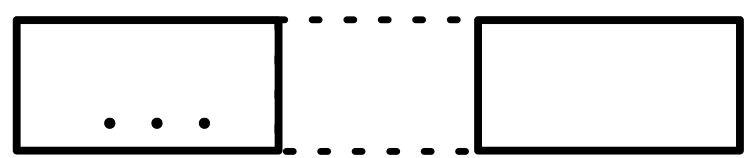
$$f_1\, v_1 \precsim^{log}_{W'} f_2\, v_2 : \tau'$$

$$W'.\mathcal{I} \;=\; \boxed{\;P_f\;} \;\cdots\; \boxed{\;P_v\;}$$

Kripke logical relation for linearity

$$f_1 \precsim^{log}_{W} f_2 : \tau \multimap \tau' \;=\;$$
$$\forall v_1, v_2 \,.\, v_1 \precsim^{log}_{W'} v_2 : \tau \;\wedge\; P_f(v_1, v_2) \implies$$
$$f_1\, v_1 \precsim^{log}_{W'} f_2\, v_2 : \tau'$$

$W.\mathcal{I} \;=$



$$!g_1 \precsim^{log}_{W} !g_2 : !(\tau \multimap \tau') \;\stackrel{\mathrm{def}}{=}\; g_1 \precsim^{log}_{W} g_2 : \tau \multimap \tau'$$

Kripke logical relation for linearity

$$f_1 \precsim_W^{log} f_2 : \tau \multimap \tau' \;=$$
$$\forall v_1, v_2 \,.\, v_1 \precsim_{W'}^{log} v_2 : \tau \;\wedge\; P_f(v_1, v_2) \implies$$
$$f_1\, v_1 \precsim_{W'}^{log} f_2\, v_2 : \tau'$$

$$W.\mathcal{I} \;=\; \boxed{\;\ldots\;} \cdots \boxed{\;P\;}$$

$$!g_1 \precsim_W^{log} !g_2 : !(\tau \multimap \tau') \;\stackrel{\text{def}}{=}\; g_1 \precsim_W^{log} g_2 : \tau \multimap \tau'$$

Kripke logical relation for linearity

$$f_1 \precsim_W^{log} f_2 : \tau \multimap \tau' \;=$$

$$\forall v_1, v_2 \,.\, v_1 \precsim_{W'}^{log} v_2 : \tau \;\wedge\; P_f(v_1, v_2) \implies$$

$$f_1\, v_1 \precsim_{W'}^{log} f_2\, v_2 : \tau'$$

$$W.\mathcal{I} \;=\; \boxed{\;\ldots\;} \cdots \boxed{P_\top}$$

$$!g_1 \precsim_W^{log} !g_2 : !(\tau \multimap \tau') \;\stackrel{\text{def}}{=}\; g_1 \precsim_W^{log} g_2 : \tau \multimap \tau'$$

$$P_\top(v_1, v_2) = \text{true}$$

Related Work

Fully abstract CPS for PCF [Laird]

- Uses game semantics proof

Logical relation for linear free theorems [Zhao et. al.]

- Open logical relation to ensure preservation of linear resources

TT logical relation for Lily [Bierman et. al.]

Conclusion

Fully abstract CPS in a language with rec. types
using **operational** proof techniques

requires $\tau \cong \forall\alpha.(\tau \to \alpha) \multimap \alpha$

which requires $f\ [\tau_k]\ k \quad \approx^{ctx} \quad k\ (f\ [\tau]\ id) : \tau_k$

Conclusion

Fully abstract CPS in a language with rec. types using **operational** proof techniques

requires $\tau \cong \forall\alpha.(\tau \to \alpha) \multimap \alpha$

which requires $f\ [\tau_k]\ k \quad \approx^{ctx} \quad k\ (f\ [\tau]\ id) : \tau_k$

Kripke logical relation that can distinguish

linearly-treated: $f : \tau_1 \multimap \tau_2$

unrestricted: $f : !(\tau_1 \multimap \tau_2)$

Why polymorphism: ST boundary semantics

$$\mathbf{v} : {\tau_1}^+ \rightarrow (\forall \alpha . ({\tau_2}^+ \rightarrow \alpha) \rightarrow \alpha)$$

$$^{\tau_1 \rightarrow \tau_2}\mathcal{ST}\,\mathbf{v} \longmapsto$$
$$\lambda \mathsf{x} : \tau_1 .\, ^{\tau_2}\mathcal{ST}(\text{let } \mathbf{z} = (\mathcal{TS}^{\tau_1}\mathsf{x})$$
$$\text{in } (\mathbf{v}\ \mathbf{z})[{\tau_2}^+]\ id)$$