A logical relation for *cbpv* with first-class execution stacks

There have been logical relations for *cbpv* that make use of stack dynamics, such as Harper's account for letcc in [7], but the execution stack is usually inaccessible in the surface language.

One approach to expose execution stacks to surface syntax has been the Enriched Effect Calculus [5], which strongly associates with linear logic and extends the typing judgment to account for contexts that may contain a computation, in order to faciliate construction of execution stacks in the style of lambda abstractions.

In constrast, this report formulates a simpler approach where execution stacks are constructed in a list-like manner from individual *frames,* which correspond exactly with execution frames in stack dynamics seen in *e.g.* [7, 8].

Syntax

We extend the standard *cbpv* with *continuation types* $A \rightsquigarrow B$, which are value types that model the execution stack, introduction forms $\cdot$ and $V :: F$, and elimination form $\mathsf{resume}(V, C)$.

Value type	X	::=	ans	answer type
		|	$U(A)$	suspension
		|	$A \rightsquigarrow B$	continuation accepting A, producing B
Computation type	A	::=	$F(X)$	computation producing a value X
		|	$X \to B$	function accepting X, returning B
Value	V	::=	$\mathsf{yes} \mid \mathsf{no}$	answers
		|	$\mathsf{susp}(C)$	suspension
		|	$\cdot$	empty continuation
		|	$V :: F$	compose continuation
Computation	C	::=	$\mathsf{ret}(V)$	return
		|	$\mathsf{letf}(C_1, x.C_2)$	bind
		|	$\mathsf{force}(V)$	force a suspension
		|	$\lambda(x.C)$	lambda abstraction
		|	$\mathsf{ap}(C, V)$	function application
		|	$\mathsf{resume}(V, C)$	restore a continuation
Frame	F	::=	$\mathsf{letf}(-, x.C_2)$	bind
		|	$\mathsf{ap}(-, V)$	application

Statics

Readers may want to specifically pay attention to rules introduced here that are not commonly seen in *cbpv i.e.* RESUME, CONT-EMPTY, CONT-LETF, CONT-APPLY.

$$\frac{x : X \in \Gamma}{\Gamma \vdash x : X}\ \textsc{var} \qquad \frac{}{\Gamma \vdash \mathsf{yes} : \mathsf{ans}}\ \textsc{yes} \qquad \frac{}{\Gamma \vdash \mathsf{no} : \mathsf{ans}}\ \textsc{no} \qquad \frac{\Gamma \vdash C : A}{\Gamma \vdash \mathsf{susp}(C) : U(A)}\ \textsc{susp}$$

$$\frac{\Gamma \vdash V : U(A)}{\Gamma \vdash \mathsf{force}(V) : A}\ \textsc{force} \qquad \frac{\Gamma \vdash V : X}{\Gamma \vdash \mathsf{ret}(V) : F(X)}\ \textsc{ret}$$

$$\frac{\Gamma \vdash C_1 : F(X) \qquad \Gamma, x : X \vdash C_2 : B}{\Gamma \vdash \mathsf{letf}(C_1, x.C_2) : B}\ \textsc{letf} \qquad \frac{\Gamma, x : X \vdash C : B}{\Gamma \vdash \lambda(x.C) : X \to B}\ \textsc{lambda}$$

$$\frac{\Gamma \vdash C : X \to B \qquad \Gamma \vdash V : X}{\Gamma \vdash \mathsf{ap}(C, V) : B}\ \textsc{apply} \qquad \frac{\Gamma \vdash V : A \rightsquigarrow B \qquad \Gamma \vdash C : A}{\Gamma \vdash \mathsf{resume}(V, C) : B}\ \textsc{resume}$$

$$\frac{}{\Gamma \vdash \cdot : A \rightsquigarrow A}\ \textsc{cont-empty} \qquad \frac{\Gamma \vdash V : B \rightsquigarrow C \qquad \Gamma, x : X \vdash C : B}{\Gamma \vdash V :: \mathsf{letf}(-, x.C) : F(X) \rightsquigarrow C}\ \textsc{cont-letf}$$

$$\frac{\Gamma \vdash V_1 : B \rightsquigarrow C \qquad \Gamma \vdash V_2 : X}{\Gamma \vdash V_1 :: \mathsf{ap}(-, V_2) : (X \to B) \rightsquigarrow C}\ \textsc{cont-apply}$$

Note that the list-like structure of the continuation type naturally admits a concatenation operation *on the meta level* when the types align; we denote this $V_1 +\!\!+ V_2$. An alternative account for the type theory could encode this in the source langauge as a computation returning a continuation ($F(A \rightsquigarrow B)$) instead.

Dynamics

We can categorize stepping rules in stack dynamics by whether they are operating on *terminal* computations (which gets stuck on empty stacks), and *non-terminal* computations (those who don't).

Stepping rules for non-terminal computations:

$$\begin{aligned} V \triangleright \mathsf{resume}(V', C) &\longmapsto V +\!\!+ V' \triangleright C \\ V \triangleright \mathsf{force}(\mathsf{susp}(C)) &\longmapsto V \triangleright C \\ V \triangleright \mathsf{letf}(C_1, x.C_2) &\longmapsto V :: \mathsf{letf}(-, x.C_2) \triangleright C_1 \\ V \triangleright \mathsf{ap}(C, V) &\longmapsto V :: \mathsf{ap}(-, V) \triangleright C \end{aligned}$$

It is worth noting that $\mathsf{resume}(V, C)$ *concatenates* onto the current execution stack, rather than replacing it.

Stepping rules for terminal computations:

$$\begin{aligned} V :: \mathsf{letf}(-, x.C_2) \triangleright \mathsf{ret}(V') &\longmapsto V \triangleright [V'/x]C_2 \\ V :: \mathsf{ap}(-, V') \triangleright \lambda(x.C) &\longmapsto V \triangleright [V'/x]C \end{aligned}$$

As is convention, we write $V \triangleright C \longmapsto^* V' \triangleright C'$ to mean the transitive closure of stepping (*i.e.* transition in a finite number of steps).

Lemma 1 (stepwise stack extension)**.** *If $V \triangleright C \longmapsto V' \triangleright C'$ then $V_0 +\!\!+ V \triangleright C \longmapsto V_0 +\!\!+ V' \triangleright C'$. Intuitively, this states that no stepping rule exhibits "action at a distance",* i.e. *dependent on frames beyond the top of the stack.*

Proof. Case on the judgment $V \triangleright C \longmapsto V' \triangleright C'$. All cases are immediate, but we spell it out just to verify.

- If $V \triangleright \mathsf{resume}(V', C) \longmapsto V +\!\!+ V' \triangleright C$, then it immediately follows that
$$V_0 +\!\!+ V \triangleright \mathsf{resume}(V', C) \longmapsto V_0 +\!\!+ V +\!\!+ V' \triangleright C.$$
- If $V \triangleright \mathsf{force}(\mathsf{susp}(C)) \longmapsto V \triangleright C$, then it immediately follows that
$$V_0 +\!\!+ V \triangleright \mathsf{force}(\mathsf{susp}(C)) \longmapsto V_0 +\!\!+ V \triangleright C.$$
- If $V \triangleright \mathsf{letf}(C_1, x.C_2) \longmapsto V :: \mathsf{letf}(-, x.C_2) \triangleright C_1$, then it immediately follows that
$$V_0 +\!\!+ V \triangleright \mathsf{letf}(C_1, x.C_2) \longmapsto V_0 +\!\!+ V :: \mathsf{letf}(-, x.C_2) \triangleright C_1.$$
- If $V :: \mathsf{letf}(-, x.C_2) \triangleright \mathsf{ret}(V') \longmapsto V \triangleright [V'/x]C_2$, then it immediately follows that
$$V_0 +\!\!+ V :: \mathsf{letf}(-, x.C_2) \triangleright \mathsf{ret}(V') \longmapsto V_0 +\!\!+ V \triangleright [V'/x]C_2.$$
- If $V :: \mathsf{ap}(-, V') \triangleright \lambda(x.C) \longmapsto V \triangleright [V'/x]C$, then it immediately follows that
$$V_0 +\!\!+ V :: \mathsf{ap}(-, V') \triangleright \lambda(x.C) \longmapsto V_0 +\!\!+ V \triangleright [V'/x]C.$$ ■

Lemma 2 (stack extension)**.** *If $V \triangleright C \longmapsto^* V' \triangleright C'$ then $V_0 +\!\!+ V \triangleright C \longmapsto^* V_0 +\!\!+ V' \triangleright C'$.*

Proof. By induction on the number of steps in the judgment $V \triangleright C \longmapsto^* V' \triangleright C'$.

- *(base case.)* Trivial.
- *(inductive case.)* Then we know that there is some smaller $V \triangleright C \longmapsto^* V'' \triangleright C''$, such that $V'' \triangleright C'' \longmapsto V' \triangleright C'$.
  - Apply IH on the first judgment to get $V_0 +\!\!+ V \triangleright C \longmapsto^* V_0 +\!\!+ V'' \triangleright C''$.
  - Invoke Lemma 1 on the second judgment to get $V_0 +\!\!+ V'' \triangleright C'' \longmapsto V_0 +\!\!+ V' \triangleright C'$.

  Compose the two to get the desired result. ■

Logical relation

We define three relations for three classes of syntactic objects: values, terminal computations, and computations in general.

For *values:*

$$\begin{aligned} V \doteq V' \in \mathsf{ans} \quad &\iff V, V' \Downarrow \mathsf{yes} \text{ or } V, V' \Downarrow \mathsf{no} \\ \in U(A) \quad &\iff V \Downarrow \mathsf{susp}(C), V' \Downarrow \mathsf{susp}(C'), C \doteq C' \in A \\ \in A \rightsquigarrow B &\iff \text{given } C \doteq C' \ \mathsf{final} \in A, V \triangleright C \downarrow V' \triangleright C' \end{aligned}$$

The definition for equivalence over continuations state that, "for equivalent terminal computations, evaluating the stacks against them again yields equivalent terminal computations."

For *terminal computations:*

$$\begin{aligned}
&\mathsf{ret}(V) \doteq \mathsf{ret}(V') \ \mathsf{final} \in F(X) \iff V \doteq V' \in X \\
&\lambda(x.C) \doteq \lambda(x.C') \ \mathsf{final} \in X \to B \iff x : X \gg C \doteq C' \in B \\
&V \triangleright C \downarrow V' \triangleright C' \iff \\
&\quad V \triangleright C \longmapsto^* \cdot \triangleright C_1, V' \triangleright C' \longmapsto^* \cdot \triangleright C_1', C_1 \doteq C_1' \ \mathsf{final}
\end{aligned}$$

$V \triangleright C \downarrow V' \triangleright C'$ is essentially stating that the evaluation of $V \triangleright C$ and $V' \triangleright C'$ *coterminate* to equivalent terminal computations.

For *computations:*

$$C \doteq C' \in A \iff \cdot \triangleright C \downarrow \cdot \triangleright C'$$

This definition of equivalence over computations merely involves evaluating in the *empty* execution stack, which might be alarming to the reader. Nonetheless, this definition is viable due to the local stepping guarantee introduced by Lemma 2 (stack extension), as we will see in the following lemmas.

We define a few shorthands to be used in the following development, all due to Harper [9]:

- $V \in X$ (resp. C, A) means $V \doteq V \in X$.
- $\gamma \doteq \gamma' \in \Gamma$ is equivalence for substitutions, *i.e.* the equivalence of values applied element-wise.
- $\Gamma \gg V \doteq V' \in X$ (resp. C, A) means that $\gamma \doteq \gamma' \in \Gamma$ implies $\hat{\gamma}V \doteq \hat{\gamma}'V' \in X$.

Corollary 3 (empty stack is equivalent to itself)**.** *For any A, we have $\cdot \doteq \cdot \in A \rightsquigarrow A$.*

Lemma 4 ($\doteq$ is a PER)**.** *All three forms of extensional equivalence $\doteq$ defined above are symmetric and transitive.*

Proof. By simultaneous induction on all three forms of extensional equivalence. ∎

Lemma 5 (cotermination)**.** *If $V \doteq V' \in A \rightsquigarrow B$ and $C \doteq C' \in A$ then $V \triangleright C \downarrow V' \triangleright C'$.*

Proof. Since $C \doteq C' \in A$, we have $\cdot \triangleright C \downarrow \cdot \triangleright C'$, namely

$$\cdot \triangleright C \longmapsto^* \cdot \triangleright C_1 \quad \text{and} \quad \cdot \triangleright C' \longmapsto^* \cdot \triangleright C_1' \quad \text{and} \quad C_1 \doteq C_1' \ \mathsf{final} \in A.$$

Applying Lemma 2 (stack extension) with V, V' respectively, we have

$$V \triangleright C \longmapsto^* V \triangleright C_1 \quad \text{and} \quad V' \triangleright C' \longmapsto^* V' \triangleright C_1' \quad \text{and} \quad C_1 \doteq C_1' \ \mathsf{final} \in A.$$

Then, since $V \doteq V' \in A \rightsquigarrow B$, we know that given $C_1 \doteq C_1' \ \mathsf{final} \in A$,

$$V \triangleright C_1 \downarrow V' \triangleright C_1'.$$

This is exactly the same as $V \triangleright C \downarrow V' \triangleright C'$. ∎

Lemma 6 (bind preserves stack equivalence)**.** *If $V \doteq V' \in A \rightsquigarrow B$, and $x : X \gg C \doteq C' : A$, then $V :: \mathsf{letf}(-, x.C) \doteq V' :: \mathsf{letf}(-, x.C') \in F(X) \rightsquigarrow B$.*

Proof. To show this, we need to consider all terminal computations $C_0 \doteq C_0' \ \mathsf{final} \in F(X)$ and show that they lead to cotermination when applied to each stack respectively.

Any such computations are of the form $\mathsf{ret}(V_0), \mathsf{ret}(V_0')$ respectively, where $V_0 \doteq V_0' \in X$. Evaluating LHS:

$$\begin{aligned} & V :: \mathsf{letf}(-, x.C) \triangleright \mathsf{ret}(V_0) \\ \longmapsto\ & V \triangleright [V_0/x]C \end{aligned}$$

Similarly, RHS evaluates to $V' \triangleright [V_0'/x]C'$.

Since $V_0 \doteq V_0' \in X$, we may invoke $x : X \gg C \doteq C' : A$ to get $[V_0/x]C \doteq [V_0'/x]C' \in A$, and then by Lemma 5 (cotermination) we have $V \triangleright [V_0/x]C \downarrow V' \triangleright [V_0'/x]C'$. ∎

Lemma 7 (application preserves stack equivalence)**.** *If $V \doteq V' \in B \rightsquigarrow C$, and $V_0 \doteq V_0' : X$, then $V :: \mathsf{ap}(-, V_0) \doteq V' :: \mathsf{ap}(-, V_0') \in (X \to B) \rightsquigarrow C$.*

Proof. To show this, we need to consider all terminal computations $C_0 \doteq C_0'\ \mathsf{final} \in X \to B$ and show that they lead to cotermination when applied to each stack respectively.

Any such computations are of the form $\lambda(x.C), \lambda(x.C')$ respectively, where $x : X \gg C \doteq C' \in B$. Evaluating LHS:

$$\begin{aligned} & V :: \mathsf{ap}(-, V_0) \triangleright \lambda(x.C) \\ \longmapsto\ & V \triangleright [V_0/x]C \end{aligned}$$

Similarly, RHS evaluates to $V' \triangleright [V_0'/x]C'$.

Since $V_0 \doteq V_0' \in X$, we may invoke $x : X \gg C \doteq C' : B$ to get $[V_0/x]C \doteq [V_0'/x]C' \in B$, and then by Lemma 5 (cotermination) we have $V \triangleright [V_0/x]C \downarrow V' \triangleright [V_0'/x]C'$. ∎

We are now equipped to demonstrate reflexivity of this logical relation.

Theorem 8 (reflexivity)**.** *If $\Gamma \vdash V : X$ then $\Gamma \gg V \in X$; if $\Gamma \vdash C : A$ then $\Gamma \gg C \in A$.*

Proof. By induction on the typing judgment.

Values.

- $\Gamma \vdash x : X$. Fix some $\gamma \doteq \gamma' \in \Gamma$, then we have $\gamma(x) \doteq \gamma'(x) \in X$.
- $\Gamma \vdash \mathsf{yes} : \mathsf{ans}$ and $\Gamma \vdash \mathsf{no} : \mathsf{ans}$. Immediate.
- $\Gamma \vdash \mathsf{susp}(C) : U(A)$. Fix some $\gamma \doteq \gamma' \in \Gamma$. By IH, $\Gamma \gg C \in A$, so $\hat{\gamma}C \doteq \hat{\gamma}'C \in A$, which is what we wanted to show.
- $\Gamma \vdash \cdot : A \rightsquigarrow A$. Follows from Corollary 3 (empty stack is equivalent to itself).
- $\Gamma \vdash V :: \mathsf{letf}(-, x.C) : F(X) \rightsquigarrow C$. Fix some $\gamma \doteq \gamma' \in \Gamma$. By IH:
  - $\Gamma \gg V \in B \rightsquigarrow C$ for some B, so $\hat{\gamma}V \doteq \hat{\gamma}'V \in B \rightsquigarrow C$.
  - $\Gamma, x : X \gg C : B$, so $x : X \gg \hat{\gamma}C \doteq \hat{\gamma}'C \in B$.
  - Thus, by Lemma 6 (bind preserves stack equivalence) we have

    $$\hat{\gamma}V :: \mathsf{letf}(-, x.\hat{\gamma}C) \doteq \hat{\gamma}'V :: \mathsf{letf}(-, x.\hat{\gamma}'C).$$

- $\Gamma \vdash V :: \mathsf{ap}(-, V') : (X \to B) \rightsquigarrow C$. Fix some $\gamma \doteq \gamma' \in \Gamma$. By IH:
  - $\Gamma \gg V \in B \rightsquigarrow C$, so $\hat{\gamma}V \doteq \hat{\gamma}'V \in B \rightsquigarrow C$.
  - $\Gamma \gg V' \in X$, so $\hat{\gamma}V' \doteq \hat{\gamma}'V' \in X$.
  - Thus, by Lemma 7 (application preserves stack equivalence) we have

    $$\hat{\gamma}V :: \mathsf{ap}(-, \hat{\gamma}V') \doteq \hat{\gamma}'V :: \mathsf{ap}(-, \hat{\gamma}'V').$$

Computations.

- $\Gamma \vdash \mathsf{ret}(V) : F(X)$. Fix some $\gamma \doteq \gamma' \in \Gamma$. Evaluating $\hat{\gamma}\ \mathsf{ret}(V)$ on the empty stack:

$$\cdot \triangleright \hat{\gamma}\ \mathsf{ret}(V) = \cdot \triangleright \mathsf{ret}(\hat{\gamma}V)\ \mathsf{final},$$

  And similarly $\cdot \triangleright \mathsf{ret}(\hat{\gamma}'V)\ \mathsf{final}$. It remains to prove that these two final states are related: By IH, $\Gamma \gg V \in X$, so $\hat{\gamma}V \doteq \hat{\gamma}'V \in X$.

- $\Gamma \vdash \mathsf{letf}(C_1, x.C_2) : B$. Fix some $\gamma \doteq \gamma' \in \Gamma$. Evaluating $\hat{\gamma}\ \mathsf{letf}(C_1, x.C_2)$ on the empty stack:

$$\begin{aligned} &\cdot \triangleright \mathsf{letf}(\hat{\gamma}C_1, x.\hat{\gamma}C_2) \\ \longmapsto\ &\cdot :: \mathsf{letf}(-, x.\hat{\gamma}C_2) \triangleright \hat{\gamma}C_1, \end{aligned}$$

  And similarly $\cdot :: \mathsf{letf}(-, x.\hat{\gamma}'C_2) \triangleright \hat{\gamma}'C_1$.

  - By IH, $\Gamma, x : X \gg C_2 \in B$ for some X. Therefore, $x : X \gg \hat{\gamma}C_2 \doteq \hat{\gamma}'C_2 \in B$. Combining Corollary 3 (empty stack is equivalent to itself) and Lemma 6 (bind preserves stack equivalence), we have

$$\cdot :: \mathsf{letf}(-, x.\hat{\gamma}C_2) \doteq \cdot :: \mathsf{letf}(-, x.\hat{\gamma}'C_2) \in F(X) \rightsquigarrow B.$$

  - Also by IH, $\Gamma \gg C_1 \in F(X)$. Therefore, $\hat{\gamma}C_1 \doteq \hat{\gamma}'C_1 \in F(X)$. Now, by Lemma 5 (cotermination),

$$\cdot :: \mathsf{letf}(-, x.\hat{\gamma}C_2) \triangleright \hat{\gamma}C_1 \downarrow \cdot :: \mathsf{letf}(-, x.\hat{\gamma}'C_2) \triangleright \hat{\gamma}'C_1,$$

    which is what we wanted to show.

- $\Gamma \vdash \mathsf{force}(V) : A$. Fix some $\gamma \doteq \gamma' \in \Gamma$. By IH, $\hat{\gamma}V \doteq \hat{\gamma}'V \in U(A)$, *i.e.* the forcing of them evaluate to related computations $C \doteq C' \in A$, which coterminate in the empty stack.

- $\Gamma \vdash \mathsf{resume}(V, C) : B$. Fix some $\gamma \doteq \gamma' \in \Gamma$. By IH, $\hat{\gamma}V \doteq \hat{\gamma}'V \in A \rightsquigarrow B$, and $\hat{\gamma}C \doteq \hat{\gamma}'C \in A$ (for some A). Evaluating in the empty stack yields:

$$\cdot \triangleright \mathsf{resume}(\hat{\gamma}V, \hat{\gamma}C) \longmapsto \hat{\gamma}V \triangleright \hat{\gamma}C,$$

  And similarly $\hat{\gamma}'V \triangleright \hat{\gamma}'C$. Direct application of Lemma 5 (cotermination) shows that they coterminate.

- $\Gamma \vdash \lambda(x.C) : X \to B$. Fix some $\gamma \doteq \gamma' \in \Gamma$. Evaluating $\hat{\gamma}\lambda(x.C)$ on the empty stack:

$$\cdot \triangleright \hat{\gamma}\lambda(x.C) = \cdot \triangleright \lambda(x.\hat{\gamma}C)\ \mathsf{final},$$

  And similarly $\cdot \triangleright \lambda(x.\hat{\gamma}'C)\ \mathsf{final}$. It remains to prove that these two final states are related: by IH, $\Gamma, x : X \gg C \in B$, so $x : X \gg \hat{\gamma}C \doteq \hat{\gamma}'C \in B$.

- $\Gamma \vdash \mathsf{ap}(C, V) : B$. Fix some $\gamma \doteq \gamma' \in \Gamma$. Evaluating $\hat{\gamma}\ \mathsf{ap}(C, V)$ on the empty stack:

$$\cdot \triangleright \hat{\gamma}\ \mathsf{ap}(C, V) \longmapsto \cdot :: \mathsf{ap}(-, \hat{\gamma}V) \triangleright \hat{\gamma}C,$$

  and similarly $\cdot :: \mathsf{ap}(-, \hat{\gamma}'V) \triangleright \hat{\gamma}'C$.

  - By IH, $\Gamma \gg V \in X$ for some X. Therefore, combining Corollary 3 (empty stack is equivalent to itself) and Lemma 7 (application preserves stack equivalence), we have

$$\cdot :: \mathsf{ap}(-, \hat{\gamma}V) \doteq \cdot :: \mathsf{ap}(-, \hat{\gamma}'V) \in (X \to B) \rightsquigarrow B.$$

- Also by IH, $\Gamma \gg C \in X \to B$. Therefore, $\hat{\gamma} C \doteq \hat{\gamma}' C \in X \to B$. Now, by Lemma 5 (cotermination),

  $$\cdot :: \mathsf{ap}(-, \hat{\gamma} V) \triangleright \hat{\gamma} C \downarrow \cdot :: \mathsf{ap}(-, \hat{\gamma}' V) \triangleright \hat{\gamma}' C,$$

  which is what we wanted to show. ∎

Equivalence laws

We write down a few equivalence laws pertaining to resume and then prove them under our logical relation.

$$\frac{\Gamma \vdash C \equiv C' : A}{\Gamma \vdash \mathsf{resume}(\cdot, C) \equiv C' : A}\ \textsc{resume-id}$$

$$\frac{\Gamma \vdash V_1 \equiv V_1' : A \rightsquigarrow B \qquad \Gamma \vdash V_2 \equiv V_2' : B \rightsquigarrow C \qquad \Gamma \vdash C_1 \equiv C_2' : A}{\Gamma \vdash \mathsf{resume}(V_2, \mathsf{resume}(V_1, C_1)) \equiv \mathsf{resume}(V_2' +\!\!+ V_1', C_1') : C}\ \textsc{resume-concat}$$

$$\frac{\Gamma \vdash V \equiv V' : B \rightsquigarrow C \qquad \Gamma, x : X \vdash C_2 \equiv C_2' : B \qquad \Gamma \vdash C_1 \equiv C_1' : F(X)}{\Gamma \vdash \mathsf{resume}(V :: \mathsf{letf}(-, x.C_2), C_1) \equiv \mathsf{resume}(V', \mathsf{letf}(C_1', x.C_2')) : C}\ \textsc{resume-letf}$$

$$\frac{\Gamma \vdash V \equiv V' : B \rightsquigarrow C \qquad \Gamma \vdash V_1 \equiv V_1' : X \qquad \Gamma \vdash C_1 \equiv C_1' : X \to B}{\Gamma \vdash \mathsf{resume}(V :: \mathsf{ap}(-, V_1), C_1) \equiv \mathsf{resume}(V', \mathsf{ap}(C_1', V_1')) : C}\ \textsc{resume-apply}$$

Lemma 9 (stack concatenation preserves equivalence). *If $V_1 \doteq V_1' \in A \rightsquigarrow B$ and $V_2 \doteq V_2' \in B \rightsquigarrow C$, then $V_2 +\!\!+ V_1 \doteq V_2' +\!\!+ V_1' \in A \rightsquigarrow C$.*

Proof. Fix some $C_0 \doteq C_0'\ \mathsf{final} \in A$. Since $V_1 \doteq V_1' \in A \rightsquigarrow B$, we know that

$$V_1 \triangleright C_0 \longmapsto^* \cdot \triangleright C_1 \quad \text{and} \quad V_1' \triangleright C_0' \longmapsto^* \cdot \triangleright C_1' \quad \text{and} \quad C_1 \doteq C_1'\ \mathsf{final} \in B.$$

Applying Lemma 2 (stack extension) with V_2, V_2', we have

$$V_2 +\!\!+ V_1 \triangleright C_0 \longmapsto^* V_2 \triangleright C_1 \quad \text{and} \quad V_2' +\!\!+ V_1' \triangleright C_0' \longmapsto^* V_2' \triangleright C_1' \quad \text{and} \quad C_1 \doteq C_1'\ \mathsf{final} \in B.$$

Then since $V_2 \doteq V_2' \in B \rightsquigarrow C$, we know that

$$V_2 \triangleright C_1 \longmapsto^* \cdot \triangleright C_2 \quad \text{and} \quad V_2' \triangleright C_1' \longmapsto^* \cdot \triangleright C_2' \quad \text{and} \quad C_2 \doteq C_2'\ \mathsf{final} \in C,$$

So indeed $V_2 +\!\!+ V_1 \triangleright C_0 \downarrow V_2' +\!\!+ V_1' \triangleright C_0'$. ∎

Theorem 10 ($\doteq$ reflects equivalence laws). *If $\Gamma \vdash C_1 \equiv C_2 : A$, then $\Gamma \gg C_1 \doteq C_2 \in A$.*

Proof. By induction on the derivation of the judgment.

- *(case RESUME-ID.)* Fix some $\gamma \doteq \gamma' \in \Gamma$. Evaluate LHS in the empty stack:

  $$\cdot \triangleright \mathsf{resume}(\cdot, \hat{\gamma} C) \longmapsto \cdot \triangleright \hat{\gamma} C',$$

  And by IH we know that $\cdot \triangleright \hat{\gamma} C \downarrow \cdot \triangleright \hat{\gamma}' C'$.

- *(case RESUME-CONCAT.)* Fix some $\gamma \doteq \gamma' \in \Gamma$. Evaluate both sides in the empty stack:

$$\begin{aligned} &\cdot \triangleright \mathsf{resume}(\hat{\gamma} V_2, \mathsf{resume}(\hat{\gamma} V_1, \hat{\gamma} C_1)) \\ &\longmapsto \hat{\gamma} V_2 \triangleright \mathsf{resume}(\hat{\gamma} V_1, \hat{\gamma} C_1) \\ &\longmapsto \hat{\gamma} V_2 +\!\!+ \hat{\gamma} V_1 \triangleright \hat{\gamma} C_1, \end{aligned} \qquad \begin{aligned} &\cdot \triangleright \mathsf{resume}(\hat{\gamma}' V_2' +\!\!+ \hat{\gamma}' V_1', \hat{\gamma}' C_1') \\ &\longmapsto \hat{\gamma}' V_2' +\!\!+ \hat{\gamma}' V_1' \triangleright \hat{\gamma}' C_1'. \end{aligned}$$

  From IH we know that $\hat{\gamma} V_2 \doteq \hat{\gamma}' V_2' \in B \rightsquigarrow C$ and $\hat{\gamma} V_1 \doteq \hat{\gamma}' V_1' \in A \rightsquigarrow B$, so applying Lemma 9 (stack concatenation preserves equivalence) we have $\hat{\gamma} V_2 +\!\!+ \hat{\gamma} V_1 \doteq \hat{\gamma}' V_2' +\!\!+ \hat{\gamma}' V_1' \in A \rightsquigarrow C$.

  We also know that $\hat{\gamma} C_1 \doteq \hat{\gamma}' C_1' \in A$ due to IH, so applying Lemma 5 (cotermination) gives us

$$\hat{\gamma} V_2 +\!\!+ \hat{\gamma} V_1 \triangleright \hat{\gamma} C_1 \downarrow \hat{\gamma}' V_2' +\!\!+ \hat{\gamma}' V_1' \triangleright \hat{\gamma}' C_1'.$$

- *(case RESUME-LETF.)* Fix some $\gamma \doteq \gamma' \in \Gamma$. Evaluating both sides in the empty stack:

$$\begin{aligned} &\cdot \triangleright \mathsf{resume}(\hat{\gamma} V :: \mathsf{letf}(-, x.\hat{\gamma} C_2), \hat{\gamma} C_1) \\ &\longmapsto \hat{\gamma} V :: \mathsf{letf}(-, x.\hat{\gamma} C_2) \triangleright \hat{\gamma} C_1, \end{aligned} \qquad \begin{aligned} &\cdot \triangleright \mathsf{resume}(\hat{\gamma}' V', \mathsf{letf}(\hat{\gamma}' C_1', x.\hat{\gamma}' C_2')) \\ &\longmapsto \hat{\gamma}' V' \triangleright \mathsf{letf}(\hat{\gamma}' C_1', x.\hat{\gamma}' C_2') \\ &\longmapsto \hat{\gamma}' V' :: \mathsf{letf}(-, x.\hat{\gamma}' C_2') \triangleright \hat{\gamma}' C_1'. \end{aligned}$$

  Now we can apply Lemma 6 (bind preserves stack equivalence) to the first two IHs show the two evaluation stacks are equivalent, and then apply Lemma 5 (cotermination) to conclude that these two executions coterminate.

- *(case RESUME-APPLY.)* Analogous to the CONT-LETF case, but with Lemma 7 (application preserves stack equivalence). ∎

Discussion, future & related work

The viability of this logical relation is entirely dependent on local stepping as observed in Lemma 1 and Lemma 2 (stack extension); this makes it unsuitable for extensions that capture or replace the entire execution stack wholesale, such as the traditional, scopeless letcc/throw operators.

On the other hand, *delimited control* operators such as shift/reset or prompt/control [4, 10] always unwinds the execution stack until a specified *delimitation frame,* and so preserves the compositional properties that our logical relation requires. These operators also underlie the implementation of effect handlers, and so prove to be highly practical. [11]

A development on our logical relation that accounts for delimited control would likely introduce a new type of stack frame $\mathsf{prompt}_A(-)$, along with a new form of terminal computation $\mathsf{control}_A(x.C, V)$ with the following dynamics,

$$\begin{aligned} V :: \mathsf{prompt}_A(-) \triangleright \mathsf{control}_A(x.C, V') &\longmapsto V \triangleright [V'/x]C \\ V :: F \triangleright \mathsf{control}_A(x.C, V') &\longmapsto V \triangleright \mathsf{control}_A(x.C, (\cdot :: F) +\!\!+ V') \end{aligned}$$

i.e. $\mathsf{control}_A$ unwinds the stack, accruing a partial continuation in the process, until it sees the corresponding prompt_A frame, where it passes the accumulated continuation to its bundled computation.

This introduces size issues specifically regarding the equivalence of terminal computations. We would like to have

$$\mathsf{control}_C(x.C_1, V_1) \doteq \mathsf{control}_C(x.C_2, V_2)\ \mathsf{final} \in B \iff$$
$$x : A \rightsquigarrow C \gg C_1 \doteq C_2 \in C \ \text{ and } \ V_1 \doteq V_2 \in A \rightsquigarrow B,$$

which refers to types A and C on RHS that are unbounded in size, thus rendering the logical relation non-well-founded. I suspect step-indexing may solve this problem, as was used by Biernacki et al. [3] in their handling of algebraic effects. The presentation of their approach was greatly simplified due to the use of Di Gianantonio and Miculan's [6] framework of *complete ordered families of equivalences (COFE),* which I was not aware of prior to completing this report.

Relatedly, Asai [2] presented a method of circumventing size issues by tracking answer types and the final program result type in type signatures, although I am unsure of the scalability of this type system. Aristizábal et al. [1] described an alternative approach to semantic equivalence in the presence of delimited control, namely via bisimulations instead of logical relations.

Bibliography

[1] Andrés Aristizábal, Dariusz Biernacki, Sergueï Lenglet, and Piotr Polesiuk. 2016. Environmental Bisimulations for Delimited-Control Operators with Dynamic Prompt Generation. In *LIPIcs (LIPIcs)*, June 2016. Porto, Portugal. https://doi.org/10.4230/LIPIcs.FSCD.2016.9

[2] Kenichi Asai. 2005. Logical relations for call-by-value delimited continuations. In *Symposium on Trends in Functional Programming*, 2005. Retrieved from https://api.semanticscholar.org/CorpusID:181956

[3] Dariusz Biernacki, Maciej Piróg, Piotr Polesiuk, and Filip Sieczkowski. 2017. Handle with care: relational interpretation of algebraic effects and handlers. *Proc. ACM Program. Lang.* 2, POPL (December 2017). https://doi.org/10.1145/3158096

[4] R. Kent Dyvbig, Simon Peyton Jones, and Amr Sabry. 2007. A monadic framework for delimited continuations. *J. Funct. Program.* 17, 6 (November 2007), 687–730. https://doi.org/10.1017/S0956796807006259

[5] Jeff Egger, Rasmus Ejlers Møgelberg, and Alex Simpson. 2014. The enriched effect calculus: syntax and semantics. *Journal of Logic and Computation* 24, 3 (2014), 615–654. https://doi.org/10.1093/logcom/exs025

[6] Pietro Di Gianantonio and Marino Miculan. 2002. A unifying approach to recursive and co-recursive definitions. In *Proceedings of the 2002 International Conference on Types for Proofs and Programs (TYPES'02)*, 2002. Springer-Verlag, Berg en Dal, The Netherlands, 148–161.

[7] Robert Harper. 2026. Effects in Call-by-Push-Value. Retrieved from https://www.cs.cmu.edu/~rwh/courses/atpl/pdfs/effects.pdf

[8] Robert Harper. 2026. Continuations, aka Contradictions, aka Contexts, aka staCks. Retrieved from https://www.cs.cmu.edu/~rwh/courses/atpl/pdfs/cont.pdf

[9] Robert Harper. 2026. Semantic Equality for Typed Lambda λ-Calculus. Retrieved from https://www.cs.cmu.edu/~rwh/courses/atpl/pdfs/semeq.pdf

[10] Chung-chieh Shan. 2004. Shift to control. In *Proceedings of the 5th workshop on Scheme and Functional Programming*, 2004. 99–107.

[11] Ningning Xie, Jonathan Immanuel Brachthäuser, Daniel Hillerström, Philipp Schuster, and Daan Leijen. 2020. Effect handlers, evidently. *Proc. ACM Program. Lang.* 4, ICFP (August 2020). https://doi.org/10.1145/3408981