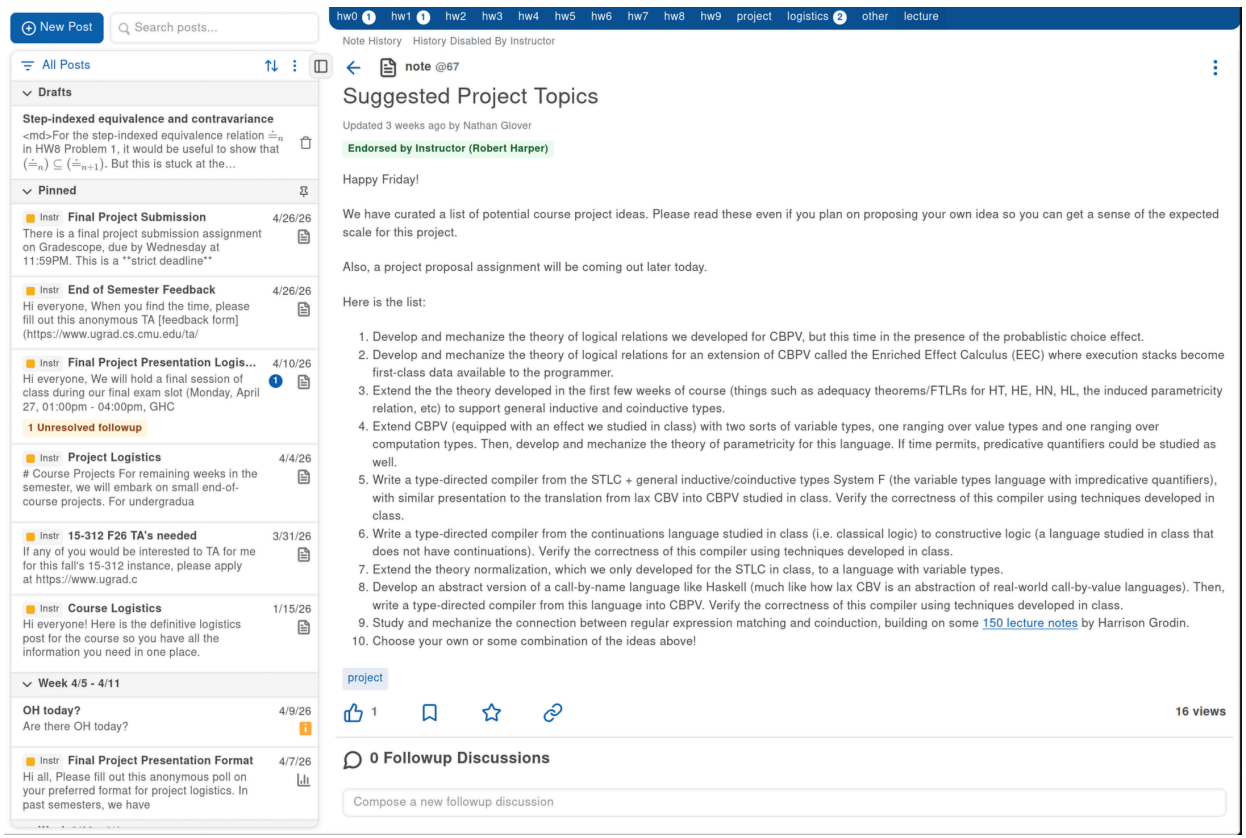
A logical relation for *cbpv* with first-class execution stacks

[redacted], 3rd year, BSCS

My thought process



My thought process

*"Develop and mechanize the theory of logical relations for an extension of CBPV called the Enriched Effect Calculus (EEC) where **execution stacks become first-class data available to the programmer**."*

An *execution stack* is some transformation $A \rightsquigarrow B$ from one computation type to another, as opposed to function types $X \to B$, which maps *values* to computations.

What is EEC? [5]

The above discussion is intended to give informal motivation for considering typing rules for the effect calculus based on two judgement forms:

$$\text{(i)}\quad \Gamma \mid - \vdash t : \mathsf{A}$$
$$\text{(ii)}\quad \Gamma \mid z : \underline{\mathsf{A}} \vdash t : \underline{\mathsf{B}}\ ,$$

where Γ is a context of value-type assignments to variables, i.e., a finite-domain partial function from variables to value types. On the right of Γ is a *stoup* (following the terminology of [13]),

~~"a *stoup*"~~ – apparently, Girard invented this terminology:

stoup *(n.)* A receptacle for holy water, especially a basin set at the entrance of a church.

What is EEC?

The resulting *enriched effect calculus* has types defined by extending the grammar for value and computation types of the effect calculus with the following additional type constructors.

$$A ::= \ldots \mid \underline{A} \multimap \underline{B} \mid !A \otimes \underline{B} \mid \underline{0} \mid \underline{A} \oplus \underline{B} \mid A \to B$$
$$\underline{A} ::= \ldots \mid !A \otimes \underline{B} \mid \underline{0} \mid \underline{A} \oplus \underline{B} \ .$$

It's heavily linear logic-flavored?

What is EEC?

$$\frac{\Gamma \mid z : \underline{\mathsf{A}} \vdash t : \underline{\mathsf{B}}}{\Gamma \mid - \vdash \lambda^{\circ} z : \underline{\mathsf{A}}.\, t : \underline{\mathsf{A}} \multimap \underline{\mathsf{B}}} \qquad \frac{\Gamma \mid - \vdash s : \underline{\mathsf{A}} \multimap \underline{\mathsf{B}} \quad \Gamma \mid \Delta \vdash t : \underline{\mathsf{A}}}{\Gamma \mid \Delta \vdash s[t] : \underline{\mathsf{B}}}$$

Stacks are constructed by lambda-like abstractions.

Maybe try something else.

Other ways to model execution stacks

An execution stack is a list of *execution frames* as seen in stack dynamics [7, 8]:

$$K \triangleright \mathsf{ap}(C, V) \longmapsto K \circ \mathsf{ap}(-, V) \triangleright C$$
$$\cdot \triangleright \mathsf{ret}(V)\ \mathsf{final}$$

How to expose this to surface syntax?

Additions to *cbpv*

Bare-minimum *cbpv* [9], equipped with first-class execution stacks:

$$
\begin{aligned}
X &::= \mathsf{ans} \mid U(A) \mid A \rightsquigarrow B \\
A &::= F(X) \mid X \to B \\
V &::= \mathsf{yes} \mid \mathsf{no} \mid \mathsf{susp}(C) \mid \cdot \mid V :: F \\
C &::= \mathsf{ret}(V) \mid \mathsf{letf}(C_1, x.C_2) \mid \mathsf{force}(V) \mid \lambda(x.C) \mid \mathsf{ap}(C, V) \\
&\quad \mid \mathsf{resume}(V, C) \\
F &::= \mathsf{letf}(-, x.C_2) \mid \mathsf{ap}(-, V)
\end{aligned}
$$

Typing stacks

$$\frac{}{\Gamma \vdash \cdot : A \rightsquigarrow A}\ \textsc{cont-empty}$$

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

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

We write $V_1 +\!\!+ V_2$ for *stack concatenation.*

Applying stacks

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

It actually returns?

Stack dynamics

$$V \triangleright \mathsf{force}(\mathsf{susp}(C)) \longmapsto V \triangleright C$$

$$V \triangleright \mathsf{letf}(C_1, x.C_2) \longmapsto V :: \mathsf{letf}(-, x.C_2) \triangleright C_1$$

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

$$V :: \mathsf{letf}(-, x.C_2) \triangleright \mathsf{ret}(V') \longmapsto V \triangleright [V'/x]C_2$$

$$V :: \mathsf{ap}(-, V') \triangleright \lambda(x.C) \longmapsto V \triangleright [V'/x]C$$

$$V \triangleright \mathsf{resume}(V', C) \longmapsto V \mathbin{\#} V' \triangleright C$$

Interesting properties

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.

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

Generalizing terminal evaluation states

With the possibility of function-producing evaluation stacks, we can't simply say "equivalent stacks produce equivalent values".

We have a set of *terminal computations*, i.e. ones that gets stuck on empty execution stacks.

$$\cdot \triangleright \mathsf{ret}(V') \ \mathsf{final}$$

$$\cdot \triangleright \lambda(x.C) \ \mathsf{final}$$

They may become unstuck once the execution stack is extended.

Generalizing terminal evaluation states

Define equivalence over *terminal computations:*

$$\mathsf{ret}(V) \doteq \mathsf{ret}(V') \ \mathsf{final} \in F(X) \quad \iff V \doteq V' \in X$$

$$\lambda(x.C) \doteq \lambda(x.C') \ \mathsf{final} \in X \to B \iff x : X \gg C \doteq C' \in B$$

Define *cotermination:*

$$V \triangleright C \downarrow V' \triangleright C' \iff$$

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

Logical relation

Size issues:

$$V \doteq V' \in A \rightsquigarrow B \iff \text{given } C \doteq C' \text{ final} \in A, V \triangleright C \downarrow V' \triangleright C'$$

$$C \doteq C' \in A \iff \text{given } V \doteq V' \in A \rightsquigarrow B, V \triangleright C \downarrow V' \triangleright C'$$

B is unbounded in size!

Second try at logical relation

$$V \doteq V' \in A \rightsquigarrow B \iff \text{given } C \doteq C' \ \mathsf{final} \in A, V \triangleright C \downarrow V' \triangleright C'$$

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

What about evaluation in nontrivial environments?

By applying Lemma 2 (stack extension):

Lemma 4 (cotermination). If $V \doteq V' \in A \rightsquigarrow B$ and $C \doteq C' \in A$ then $V \triangleright C \downarrow V' \triangleright C'$.

This allows us to prove:

Lemma 5 (bind preserves stack equivalence). If $V \doteq V' \in A \rightsquigarrow B$, and $x : X \gg C \doteq C' : A$, then $V :: \mathsf{letf}(-, x.C) \doteq V' :: \mathsf{letf}(-, x.C') \in F(X) \rightsquigarrow B$.

Lemma 6 (application preserves stack equivalence). If $V \doteq V' \in B \rightsquigarrow C$, and $V_0 \doteq V_0' : X$, then $V :: \mathsf{ap}(-, V_0) \doteq V' :: \mathsf{ap}(-, V_0') \in (X \to B) \rightsquigarrow C$.

Theorem 7 (reflexivity). If $\Gamma \vdash V : X$ then $\Gamma \gg V \in X$; if $\Gamma \vdash C : A$ then $\Gamma \gg C \in A$.

That's it

I ran out of time to prove equivalence laws.

I also lied at the beginning of the presentation.

A *failed attempt* at logical relation for *cbpv* with delimited continuations

[redacted], 3rd year, BSCS

What are delimited continuations

Traditional control operators (letcc/throw) capture/replace the stack wholesale. Action at a distance!

Delimited control operators (shift/reset or prompt/control) [4, 10] always unwinds the stack to a specific *delimiter frame.*

$$V :: \mathsf{prompt}_A(-) \triangleright \mathsf{control}_A(x.C, V') \longmapsto V \triangleright [V'/x]C$$

$$V :: F \triangleright \mathsf{control}_A(x.C, V') \longmapsto V \triangleright \mathsf{control}_A(x.C, [F] +\!\!+ V')$$

Where it breaks down

A control that hasn't found its prompt should be considered terminal:

$$\cdot \triangleright \mathsf{control}_A(x.C, V)\ \mathsf{final}$$

But for the logical relation to work, we want:

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

A, C are again unbounded in size.

What to do next

We probably need step-indexing: as Biernacki et al. [3] did in their treatment of algebraic effects.

Di Gianantonio and Miculan's [6] framework of *complete ordered families of equivalences* may be a useful tool.

Asai [2] circumvents size issues by tracking all relevant types in type signatures, which seems impractical.

Aristizábal et al. [1] provides an account for delimited control using bisimulations.

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. Continuations, aka Contradictions, aka Contexts, aka staCks. Retrieved from https://www.cs.cmu.edu/~rwh/courses/atpl/pdfs/cont.pdf

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

[9] Robert Harper. 2026. Polarization: Call-by-Push-Value. Retrieved from https://www.cs.cmu.edu/~rwh/courses/atpl/pdfs/cbpv.pdf

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

Questions and (hopefully) answers

https://dayli.ly/scrapbox/cbpv-stacks.pdf