

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

██████████, 3rd year, BSCS

# My thought process

The screenshot shows a forum interface with a left sidebar and a main content area. The sidebar contains a list of posts with titles like 'Step-indexed equivalence and contravariance', 'Final Project Submission', 'End of Semester Feedback', 'Final Project Presentation Logis...', 'Project Logistics', '15-312 F26 TA's needed', and 'Course Logistics'. The main content area shows a post by 'note @67' titled 'Suggested Project Topics', updated 3 weeks ago by Nathan Glover. The post is endorsed by Robert Harper and contains a list of 10 project topics. The interface includes navigation tabs at the top, a search bar, and interaction buttons like 'Like', 'Bookmark', 'Star', and 'Share'.

**Suggested Project Topics**  
Updated 3 weeks ago by Nathan Glover  
Endorsed by Instructor (Robert Harper)

Happy Friday!

We have curated a list of potential course project ideas. Please read these even if you plan on proposing your own idea so you can get a sense of the expected scale for this project.

Also, a project proposal assignment will be coming out later today.

Here is the list:

1. Develop and mechanize the theory of logical relations we developed for CBPV, but this time in the presence of the probabilistic choice effect.
2. 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.
3. Extend the the theory developed in the first few weeks of course (things such as adequacy theorems/FTLRs for HT, HE, HN, HL, the induced parametricity relation, etc) to support general inductive and coinductive types.
4. Extend CBPV (equipped with an effect we studied in class) with two sorts of variable types, one ranging over value types and one ranging over computation types. Then, develop and mechanize the theory of parametricity for this language. If time permits, predicative quantifiers could be studied as well.
5. Write a type-directed compiler from the STLC + general inductive/coinductive types System F (the variable types language with impredicative quantifiers), with similar presentation to the translation from lax CBV into CBPV studied in class. Verify the correctness of this compiler using techniques developed in class.
6. Write a type-directed compiler from the continuations language studied in class (i.e. classical logic) to constructive logic (a language studied in class that does not have continuations). Verify the correctness of this compiler using techniques developed in class.
7. Extend the theory normalization, which we only developed for the STLC in class, to a language with variable types.
8. Develop an abstract version of a call-by-name language like Haskell (much like how lax CBV is an abstraction of real-world call-by-value languages). Then, write a type-directed compiler from this language into CBPV. Verify the correctness of this compiler using techniques developed in class.
9. Study and mechanize the connection between regular expression matching and coinduction, building on some [150 lecture notes](#) by Harrison Grodin.
10. Choose your own or some combination of the ideas above!

16 views

0 Followup Discussions

## 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  $X \rightarrow B$ , 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:

$$(i) \Gamma \mid - \vdash t : A$$

$$(ii) \Gamma \mid z : \underline{A} \vdash t : \underline{B} ,$$

where  $\Gamma$  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  $\Gamma$  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.

$$\begin{aligned} A &::= \dots \mid \underline{A} \multimap \underline{B} \mid !A \otimes \underline{B} \mid \underline{0} \mid \underline{A} \oplus \underline{B} \mid A \rightarrow B \\ \underline{A} &::= \dots \mid !A \otimes \underline{B} \mid \underline{0} \mid \underline{A} \oplus \underline{B} . \end{aligned}$$

It's heavily linear logic-flavored?

## What is EEC?

$$\frac{\Gamma \mid z:\underline{A} \vdash t:\underline{B}}{\Gamma \mid - \vdash \lambda^\circ z:\underline{A}. t:\underline{A} \multimap \underline{B}}$$

$$\frac{\Gamma \mid - \vdash s:\underline{A} \multimap \underline{B} \quad \Gamma \mid \Delta \vdash t:\underline{A}}{\Gamma \mid \Delta \vdash s[t]:\underline{B}}$$

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]:

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

How to expose this to surface syntax?

## Additions to *cbpv*

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

$$X ::= \text{ans} \mid U(A) \mid A \rightsquigarrow B$$
$$A ::= F(X) \mid X \rightarrow B$$
$$V ::= \text{yes} \mid \text{no} \mid \text{susp}(C) \mid \cdot \mid V :: F$$
$$C ::= \text{ret}(V) \mid \text{letf}(C_1, x.C_2) \mid \text{force}(V) \mid \lambda(x.C) \mid \text{ap}(C, V) \\ \mid \text{resume}(V, C)$$
$$F ::= \text{letf}(-, x.C_2) \mid \text{ap}(-, V)$$

# Typing stacks

$$\frac{}{\Gamma \vdash \cdot : A \rightsquigarrow A} \text{CONT-EMPTY}$$

$$\frac{\Gamma \vdash V : B \rightsquigarrow C \quad \Gamma, x : X \vdash C : B}{\Gamma \vdash V :: \text{letf}(-, x.C) : F(X) \rightsquigarrow C} \text{CONT-LETF}$$

$$\frac{\Gamma \vdash V_1 : B \rightsquigarrow C \quad \Gamma \vdash V_2 : X}{\Gamma \vdash V_1 :: \text{ap}(-, V_2) : (X \rightarrow B) \rightsquigarrow C} \text{CONT-APPLY}$$

We write  $V_1 \# V_2$  for *stack concatenation*.

## Applying stacks

$$\frac{\Gamma \vdash V : A \rightsquigarrow B \quad \Gamma \vdash C : A}{\Gamma \vdash \text{resume}(V, C) : B} \text{RESUME}$$

It actually returns?

## Stack dynamics

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

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

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

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

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

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

## Interesting properties

**Lemma 1** (stepwise stack extension). If  $V \triangleright C \mapsto V' \triangleright C'$  then  $V_0 \# V \triangleright C \mapsto V_0 \# V' \triangleright C'$ .

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  $V \triangleright C \mapsto^* V' \triangleright C'$  then  $V_0 \# V \triangleright C \mapsto^* V_0 \# V' \triangleright C'$ .

## 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.

- $\triangleright \text{ret}(V')$  final
- $\triangleright \lambda(x.C)$  final

They may become unstuck once the execution stack is extended.

## Generalizing terminal evaluation states

Define equivalence over *terminal computations*:

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

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

Define *cotermination*:

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

$$V \triangleright C \mapsto^* \cdot \triangleright C_1 \text{ and } V' \triangleright C' \mapsto^* \cdot \triangleright C'_1 \text{ and } C_1 \doteq C'_1 \text{ final}$$

# Logical relation

Size issues:

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

$B$  is unbounded in size!

## Second try at logical relation

$$\begin{aligned} V \doteq V' \in A \rightsquigarrow B &\iff \text{given } C \doteq C' \text{ final} \in A, V \triangleright C \downarrow V' \triangleright C' \\ C \doteq C' \in A &\iff \cdot \triangleright C \downarrow \cdot \triangleright C' \end{aligned}$$

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  $V :: \text{letf}(-, x.C) \doteq V' :: \text{letf}(-, x.C') \in F(X) \rightsquigarrow B$ .

**Lemma 6** (application preserves stack equivalence). If  $V \doteq V' \in B \rightsquigarrow C$ , and  $V_0 \doteq V'_0 : X$ , then  $V :: \text{ap}(-, V_0) \doteq V' :: \text{ap}(-, V'_0) \in (X \rightarrow B) \rightsquigarrow C$ .

**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

, 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*.

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

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

## Where it breaks down

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

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

But for the logical relation to work, we want:

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

$A, C$  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>