CIS 670 Project Proposal

Paul He, Irene Yoon

Spring 2021

1 Project Goal

In this project, we aim to perform logical relations proofs for the call-by-push-value (CBPV) calculus in the Iris proof framework. CBPV is a nifty calculus that subsumes both call-by-value (CBV) and call-by-name (CBN) calculus. This project will formalize in Coq a unary logical relation on a basic CBPV calculus and prove weak normalization and state a semantic typing relation. We plan to use the Iris framework for this work. For the scope of the current proposal, we will be using basic features of the Iris framework to define this logical relation.

We view this project as necessary groundwork for exploring an equational theory of a CBPV calculus with recursive types. Since Iris provides support for step-indexed logical relations to model recursion, the future work will be able to take further advantage of the framework.

2 Introduction

Our first step will be to define a unary logical relation for CBPV without recursive types. This will require us to learn more about Iris and using the Iris proof mode in Coq to define logical relations in the Iris logic. Once this is done, we will extend our language and logical relations to add recursive types. At this point, the support for step-indexing in Iris should simplify our definitions compared to manual step-indexing. Time permitting, we can also extend to more complex logical relations such as binary relations for contexual equivalence. Another possible future direction is to extend our language with more features that are well-suited to the features of Iris, such as higher order state or concurrency.

Formalized Logical Relations of CBPV This is primarily inspired by the pioneering work of Rizkallah et al. (2018) of a formalized equational theory of CBPV, which formalizes the soundness of the equational theory with respect to an observational equivalence. Forster et al. (2019)'s approach offers a slightly different proof setup: they formalize contexts as first-class objects using a Kripke semantics. We wish to develop a formalization similar to Forster et al. (2019) because the Iris framework offers a natural bridge to building Kripke logical relations.

Iris Proof Framework The Iris proof framework is a state-of-the-art tool that has a rich program logic that can be used for reasoning about concurrent programs. We plan to use its underlying step-indexed logic to define logical relations for CBPV. Iris's base logic is equipped with modalities that make such definition easier, especially in the presence of recursive types. Krebbers et al. (2017) presents a semantic typing result like the one we plan to do for a similar calculus with recursive types. Another similar approach is Timany et al. (2017)'s work on defining a logical relation for the state monad, where their relation induces a definition of contextual equivalence on effectful computations encapsulated by the ST monad.

3 CBPV Calculus

$$\begin{array}{rrl} \textit{value types } A & ::= & U\,\underline{C} \mid 1 \mid A_1 \times A_2 \mid A_1 + A_1 \\ \textit{computation types } \underline{C} & ::= & F\,A \mid A \to \underline{C} \mid \top \mid \underline{C_1} \,\&\, \underline{C_2} \\ \textit{environments } \Gamma & ::= & x_1 : A_1, ..., x_n : A_n \end{array}$$

$$\begin{array}{rrl} \textit{value terms } V & ::= & \mathsf{thunk}\ M \mid x \mid () \mid (V_1,\, V_2) \mid \mathsf{inj}_i\ V \\ \textit{computation terms } M & ::= & * \mid \mathsf{return}\ V \mid \lambda x.M \mid M\ V \mid \langle M_1, M_2 \rangle \mid \mathsf{force}\ V \mid \mathsf{case}(V,\, x.M_1,\, x.M_2) \\ & & \mathsf{let}\ x \leftarrow M\ \mathsf{in}\ N \mid \mathsf{prj}_i\ M \end{array}$$

Value Typing $\boxed{\Gamma \vdash^V V : A}$

$$\frac{(x:A) \in \Gamma}{\Gamma \vdash^V x : A}\,\text{VAR} \qquad \frac{}{\Gamma \vdash^V () : 1}\,\text{UNIT} \qquad \frac{\Gamma \vdash V_1 : A_1 \ \Gamma \vdash V_2 : A_2}{\Gamma \vdash^V (V_1,\, V_2) : A_1 \times A_2}\,\text{PROD} \qquad \frac{\Gamma \vdash V : A_i}{\Gamma \vdash^V \mathsf{inj}_i\ V : A_1 + A_2}\,\text{SUM} \qquad \frac{\Gamma \vdash M : \underline{C}}{\Gamma \vdash^V \mathsf{thunk}\ V : U\,\underline{C}}\,\text{THUNK}$$

Computation Typing $\boxed{\Gamma \vdash^C M : \underline{C}}$

$$\frac{}{\Gamma \vdash^C * : \top}\,\text{UNIT} \qquad \frac{\Gamma \vdash^V V : A}{\Gamma \vdash^C \mathsf{return}\ V : F\,A}\,\text{RET} \qquad \frac{\Gamma \vdash^C M : F\,A \qquad \Gamma, x : A \vdash^C N : \underline{C}}{\Gamma \vdash^C \mathsf{let} \leftarrow M\ in\ N : \underline{C}}\,\text{BIND} \qquad \frac{\Gamma, x : A \vdash^C M : \underline{C}}{\Gamma \vdash^C \lambda x.M : A \to \underline{C}}\,\text{LAM}$$

$$\frac{\Gamma \vdash^C M : A \to \underline{C} \qquad \Gamma \vdash^V V : A}{\Gamma \vdash^C M\ V : \underline{C}}\,\text{APP} \qquad \frac{\Gamma \vdash^V U\,\underline{C}}{\Gamma \vdash^C \mathsf{force}\ V : \underline{C}}\,\text{FORCE}$$

$$\frac{\Gamma \vdash^V V : A_1 + A_2 \qquad \Gamma, x_1 : A_1 \vdash^C M_1 : \underline{C} \qquad \Gamma, x_2 : A_2 \vdash^C M_2 : \underline{C}}{\Gamma \vdash^C \mathsf{case}(V,\, x_1.M_1,\, x_2.M_2) : \underline{C}}\,\text{CASE} \qquad \frac{\Gamma \vdash^C M_1 : \underline{C_1} \qquad \Gamma \vdash^C M_2 : \underline{C_2}}{\Gamma \vdash^C \langle M_1, M_2 \rangle : \underline{C_1} \,\&\, \underline{C_2}}\,\text{PAIR}$$

$$\frac{\Gamma \vdash^C \underline{C_1} \,\&\, \underline{C_2}}{\Gamma \vdash^C \mathsf{prj}_i M : \underline{C_i}}\,\text{PRJ}$$

Figure 1: CBPV Syntax and Statics

In this section, we briefly discuss the CBPV calculus that we will formalize, which is based on the calculus of Forster et al. (2019). This presentation of the CBPV calculus is a standard presentation based on a simply typed lambda calculus with binary product and sum type. For the sake of concreteness, we elaborate the syntax and static typing rules of the calculus in Figure 1.

The mantra of the CBPV calculus is "A value <u>is</u>, a computation <u>does</u>", and this distinction is made at the syntactic level. Types and terms are separated into *value* types and terms, and *computation* types and terms. All variables in the context Γ have value type. A computation in $F\,A$ aims to return a value in A, and has a return and let operator which behaves similarly to a monadic bind and return operator. A computation in $U\,\underline{C}$ is a thunk of a computation, and has two associated operators thunk and force.

The operational semantics we will formalize is a weak reduction, where a primitive reduction is defined for only for the following contexts: $C := \mathsf{let}\ x \leftarrow [] \ \mathsf{in}\ N \mid\ []\ V \mid \mathsf{prj}_i\ []$. This will allow us to prove a weak normalization and state a semantic typing relation for this simple CBPV calculus.

References

Yannick Forster, Steven Schäfer, Simon Spies, and Kathrin Stark. 2019. Call-by-Push-Value in Coq: Operational, Equational, and Denotational Theory. In *Proceedings of the 8th ACM SIGPLAN International Conference on Certified Programs and Proofs* (Cascais, Portugal) *(CPP 2019)*. Association for Computing Machinery, New York, NY, USA, 118–131. `https://doi.org/10.1145/3293880.3294097`

Robbert Krebbers, Amin Timany, and Lars Birkedal. 2017. Interactive proofs in higher-order concurrent separation logic. In *Proceedings of the 44th ACM SIGPLAN Symposium on Principles of Programming Languages*. 205–217.

Christine Rizkallah, Dmitri Garbuzov, and Steve Zdancewic. 2018. A Formal Equational Theory for Call-By-Push-Value. In *Interactive Theorem Proving*, Jeremy Avigad and Assia Mahboubi (Eds.). Springer International Publishing, Cham, 523–541.

Amin Timany, Léo Stefanesco, Morten Krogh-Jespersen, and Lars Birkedal. 2017. A Logical Relation for Monadic Encapsulation of State: Proving Contextual Equivalences in the Presence of RunST. *Proc. ACM Program. Lang.* 2, POPL, Article 64 (Dec. 2017), 28 pages. `https://doi.org/10.1145/3158152`