Monadic Realizability for Intuitionistic Higher-Order Logic

Liron Cohen[1]*, Ariel Grunfeld[1]*, and Ross Tate

Ben-Gurion University, Israel

Abstract

The standard construction for realizability semantics of intuitionistic higher-order logic is based on partial combinatory algebras as an abstract computation model with a single computational effect, namely, non-termination. Many computational effects can be modelled using monads, where programs are interpreted as morphisms in the corresponding Kleisli category. To account for a more general notion of computational effects, we here construct effectful realizability models via evidenced frames, where the underlying computational model is defined in terms of an arbitrary monad. Concretely, we generalize partial combinatory algebras to combinatory algebras over a monad and use monotonic post-modules to relate predicates to computations.

Evidenced Frames (EF) provide a general framework for constructing realizability triposes, including various computational effects that go beyond partial computation [4]. An evidenced frame is a tagged variant of complete Heyting prealgebras, where instead of the binary preorder relation $\varphi \leq \psi$ we have a ternary evidence relation $\varphi \xrightarrow{e} \psi$ where e is considered as evidence for the judgment $\varphi \leq \psi$. Similarly, each of the components of a complete Heyting prealgebra (reflexivity, transitivity, top, conjunction, implication, and universal quantification) are defined in terms of their evidence.

The standard construction of realizability models for intuitionistic higher-order logic (**iHOL**) interprets formulas as functions to subsets of the set of codes in a partial combinatory algebra (PCA), constructing a tripos (called "the realizability tripos") by using the codes as evidence for the validity of entailments, and using the functional completeness of the PCA to construct specific codes to realize the logical constants of **iHOL**. Evidenced frames can similarly be used to construct a tripos, in a manner that separates the realizability construction into two phases: first, constructing an EF from a PCA, and then constructing a tripos from the EF. This separation gives us a single structure that explicitly relates the logical content to the computational content, and allows us to replace the PCA with other viable models, such as relational combinatory algebras (RCAs) and stateful combinatory algebras (SCAs), as described in [3].

The main goal of this work is to generalize these results further by abstracting the details of the specific computational effects, and instead relying on the ideas first introduced in [9] where the effects are encapsulated behind some arbitrary monad. For this, RCA and SCA can be considered as special cases of a more general notion: Monadic Combinatory Algebra (MCA).

Definition 1 (Monadic Combinatory Algebra)**.** *Given a strong monad $T : \mathbb{C} \to \mathbb{C}$, a* Monadic Applicative Structure *(MAS) is an object of "codes" $\mathbb{A}$ together with an application Kleisli morphism: $\alpha \in \mathbb{C}(\mathbb{A} \times \mathbb{A}, T\mathbb{A})$. We say that $\mathbb{A}$ is a* Monadic Combinatory Algebra *(MCA) when $\mathbb{A}$ is a Turing object [2] in the Kleisli category of T, considered as a restriction category.*

When $\mathbb{C} = \mathbf{Set}$, the definition coincides with a PCA for T the sub-singleton monad, with an RCA for the power-set monad, and an SCA for the (increasing) state power-set monad.

In **Set**, an MCA can be more explicitly defined using an abstraction operator over terms. Given a *MAS* $\mathbb{A}$, the set $E_n(\mathbb{A})$ of *terms over* $\mathbb{A}$ is defined by the grammar:

$$e \ ::= \ 0 \mid \ \ldots \ \mid n-1 \mid c \in \mathbb{A} \mid e \bullet e$$

*This research is supported by Grant No. 2020145 from the United States-Israel Binational Science Foundation (BSF)

$E_0(\mathbb{A})$ is the set of *closed terms over* $\mathbb{A}$, and $e[c]$ denotes the straightforward *substitution*. *Evaluation* $\nu : E_0(\mathbb{A}) \to T(\mathbb{A})$ is defined by induction on $E_0(\mathbb{A})$ (using do-notation):

$$\nu(c) := \eta(c) \qquad \nu(e_f \bullet e_a) := \text{do } c_f \leftarrow \nu(e_f) \ ; \ c_a \leftarrow \nu(e_a) \ ; \ \alpha(c_f, c_a)$$

Proposition 1 (Monadic Combinatory Algebra)**.** *Given a MAS* $\mathbb{A}$*,* $\mathbb{A}$ *is an MCA if for each* $n \in \mathbb{N}$ *there's an abstraction operator* $\langle \lambda^n.(-) \rangle : E_{n+1}(\mathbb{A}) \to \mathbb{A}$ *s.t:* $\left\langle \lambda^{n+1}.e \right\rangle \cdot c = \eta(\langle \lambda^n.e[c] \rangle)$ *and* $\left\langle \lambda^0.e \right\rangle \cdot c = \nu(e[c])$*.*

MCAs allow us to construct evidence for entailments in a similar manner to PCAs. For PCAs, we consider predicates over codes and say $\varphi \xrightarrow{e} \psi$ whenever the predicate ψ is satisfied by the output of the application of e on an input which satisfies φ. In MCAs, while the input of the application is a code, the output is wrapped inside T, so to relate a predicate over codes to a predicate over wrapped codes, we need to use a *post-module* to "lift" predicates on $\mathbb{C}$ to predicates on the Kleisli category of T.

Definition 2 (post-module)**.** *Given a monad* T*, a* post-module *is a tuple* $(\mathbb{P}, P, \rho)$ *where* $\mathbb{P}$ *is a category,* $P : \mathbb{C} \to \mathbb{P}$ *is a functor, and* $\rho : PT \Rightarrow P$ *is a natural transformation for which* $\rho \circ P\eta = id_P$ *and* $\rho \circ P\mu = \rho \circ \rho_T$*.*

When $\mathbb{P} = \mathbf{Prost}^{\mathrm{op}}$ (the dual of the category of preordered sets), for any morphism f in $\mathbb{C}$, the function Pf has to be a monotonic, and similarly each component $\rho_X : PX \to PTX$ of ρ has to be monotonic. So when $\mathbb{P} = \mathbf{Prost}^{\mathrm{op}}$ we use the term *monotonic post-module*.

Proposition 2. *Given a post-module* $(\mathbb{P}, P, \rho)$*, we get a functor* $P_\rho : \mathbb{C}_T \to \mathbb{P}$ *where* $\mathbb{C}_T$ *is the Kleisli category of* T *on* $\mathbb{C}$*.* P_ρ *is defined on objects by* $P_\rho X = PX$ *and on a morphism* $f \in \mathbb{C}_T(A, B)$ *by* $P_\rho f = \rho_B \circ Pf$*.*

P_ρ can be considered as a categorical counterpart of Dijkstra's weakest precondition transformer [5] Given an MCA $\mathbb{A}$ and a *monotonic post-module* $(\mathbf{Prost}^{\mathrm{op}}, P, \rho)$ we can define an evidence relation on predicates on $\mathbb{A}$. If $\varphi, \psi \in P(\mathbb{A})$ and $e \in \mathbb{C}(\mathbf{1}, \mathbb{A})$ we say that $\varphi \xrightarrow{e} \psi$ whenever $\varphi \leq P_\rho(\alpha \circ \langle e \circ !, \mathrm{id}_{\mathbb{A}} \rangle)(\psi)$ (where ! is the unique morphism to the terminal object).

An interesting special case is when $\mathbb{C} = \mathbf{Set}$. In $\mathbf{Set}$, whenever Ω is a preordered set, we have the $\mathbf{Proset}$ functor $PA = A \to \Omega$, with $Pf(\varphi) = \varphi \circ f$. This functor can become a monotonic post-module by using a *monotonic* T*-algebra* [1] $\omega : T\Omega \to \Omega$, yielding $\rho_A(\varphi) = \omega \circ T\varphi$ (for $\varphi : A \to \Omega$). Equivalently, we can use a monad morphism into the monotone continuation monad, as described in [8]. We already have a construction (formalized in Coq) of an EF based on it. While capturing PCAs, RCAs, and SCAs, it seems not general enough to encompass other interesting EFs, such as EFs of probabilistic computation, which would probably require T to be the Giry monad [6]. Furthermore, it does not seem to encompass interesting post-modules for the continuation monad.

To get a more bird's-eye view of the necessary components needed to construct an EF, we define a deductive system called **EffHOL** which extends **iHOL** with *effectful* terms along with the operations: `return` to turn pure terms into effectful terms, `bind` to compose effectful terms, and `after` to relate effectful terms with formulas. The `after` operation acts as a quantifier that takes an effectful term e and a formula φ and returns the formula $\texttt{after}\ x := e\ .\ \varphi$, denoting that after the computation of e, the formula φ holds. To relate **EffHOL** to **iHOL**, we require **EffHOL** to have a special type of "codes" $\mathbb{A}$, denoting programs in an untyped programming language. $\mathbb{A}$ is required to have an effectful "application" operator `ap` which takes two pure codes e and c and returns an effectful term $\texttt{ap}(e, c)$, corresponding to the application of the program denoted by e to the input denoted by c, yielding a computation (hence the result is an effectful term). By combining `ap` with `after` we can use **EffHOL** to describe Hoare triples $\{\varphi\}\ e\ \{\psi\}$ [7]:

$$c_a \mid \varphi[c_a] \ \vdash \ \texttt{after}\ c_r := \texttt{ap}(e, c_a)\ .\ \psi[c_r]$$

References

[1] Alejandro Aguirre and Shin-ya Katsumata. Weakest preconditions in fibrations. *Electronic Notes in Theoretical Computer Science*, 352:5–27, 2020.

[2] J Robin B Cockett and Pieter JW Hofstra. Introduction to turing categories. *Annals of pure and applied logic*, 156(2-3):183–209, 2008.

[3] Liron Cohen, Sofia Abreu Faro, and Ross Tate. The effects of effects on constructivism. *Electronic Notes in Theoretical Computer Science*, 347:87–120, 2019.

[4] Liron Cohen, Étienne Miquey, and Ross Tate. Evidenced frames: a unifying framework broadening realizability models. In *2021 36th Annual ACM/IEEE Symposium on Logic in Computer Science (LICS)*, pages 1–13. IEEE, 2021.

[5] Edsger W Dijkstra. Guarded commands, nondeterminacy and formal derivation of programs. *Communications of the ACM*, 18(8):453–457, 1975.

[6] Michele Giry. A categorical approach to probability theory. In *Categorical Aspects of Topology and Analysis: Proceedings of an International Conference Held at Carleton University, Ottawa, August 11–15, 1981*, pages 68–85. Springer, 2006.

[7] Charles Antony Richard Hoare. An axiomatic basis for computer programming. *Communications of the ACM*, 12(10):576–580, 1969.

[8] Kenji Maillard, Danel Ahman, Robert Atkey, Guido Martínez, Catalin Hritcu, Exequiel Rivas, and Éric Tanter. Dijkstra monads for all. *arXiv preprint arXiv:1903.01237*, 2019.

[9] Eugenio Moggi. Notions of computation and monads. *Information and computation*, 93(1):55–92, 1991.