From Partial to Monadic: Combinatory Algebra with Effects

Liron Cohen
Ben-Gurion University, Israel

Ariel Grunfeld
Ben-Gurion University, Israel

Dominik Kirst
Ben-Gurion University, Israel / Inria Paris, France

Étienne Miquey
Aix Marseille University, CNRS, I2M, Marseille, France

Abstract

Partial Combinatory Algebras (PCAs) provide a foundational model of the untyped λ-calculus and serve as the basis for many notions of computability, such as realizability theory. However, PCAs support a very limited notion of computation by only incorporating non-termination as a computational effect. To provide a framework that better internalizes a wide range of computational effects, this paper puts forward the notion of Monadic Combinatory Algebras (MCAs). MCAs generalize the notion of PCAs by structuring the combinatory algebra over an underlying computational effect, embodied by a monad. We show that MCAs can support various side effects through the underlying monad, such as non-determinism, stateful computation and continuations. We further obtain a categorical characterization of MCAs within Freyd Categories, following a similar connection for PCAs. Moreover, we explore the application of MCAs in realizability theory, presenting constructions of effectful realizability triposes and assemblies derived through evidenced frames, thereby generalizing traditional PCA-based realizability semantics. The monadic generalization of the foundational notion of PCAs provides a comprehensive and powerful framework for *internally* reasoning about effectful computations, paving the path to a more encompassing study of computation and its relationship with realizability models and programming languages.

2012 ACM Subject Classification Theory of computation

Keywords and phrases Combinatory algebras, Monads, Effects, Realizability, Evidenced frames

Supplementary Material Rocq Formalization
Other (Source Code): https://github.com/dominik-kirst/mca

Funding This work was partially supported by Grant No. 2020145 from the United States-Israel Binational Science Foundation (BSF). Dominik Kirst received funding from the European Union's Horizon research and innovation programme under the Marie Skłodowska-Curie grant agreement No.101152583 and a Minerva Fellowship of the Minerva Stiftung Gesellschaft für die Forschung mbH.

Acknowledgements We are deeply grateful to Ross Tate for his valuable ideas and insights, which significantly shaped the direction of this work.

1 Introduction

Partial Combinatory Algebras (PCAs) offer a foundational algebraic model for the untyped λ-calculus, which underpins various notions of computability [12, 19]. PCAs have been widely applied in areas such as realizability interpretations of logic and type theory [21, 44], as well as in the construction of categories of assemblies and sheaves [13]. A key feature of PCAs is their partiality, meaning that the application of a program to an input is defined as a partial operator, i.e. not all inputs are guaranteed to produce valid outputs. This property allows

PCAs to naturally support non-termination as a computational effect. However, PCAs are inherently limited in their ability to model other critical effects, such as nondeterminism, stateful computation, or exceptions. Thus, despite their foundational significance, traditional PCAs are inherently limited in their computational scope and thus restricted in their utility to support modern computation, which often demands a broader treatment of side effects.

While extensions of PCAs, such as PCAs with errors (PCAE), with choice (PCAC), with partiality and exceptions (PCAX) [8, 10], have been developed to accommodate specific effects, these approaches often rely on incorporating additional structures or operations into the underlying algebraic framework. However, the ability to internally support various computational effects is crucial for the development of robust and expressive computational theories. Therefore, our goal is to develop a unified framework that internalizes a wide range of computational effects into combinatory algebra in a manner as natural as the support for non-termination in PCAs. This will allow for a more encompassing study of computation and new insights in the wide range of application domains of combinatory algebras.

A common categorical device for analyzing computational effects in the study of programming languages is through monads [28, 46]. Concretely, (strong) monads are a special kind of functors that allow the composition of Kleisli morphisms, i.e., morphisms where the target is an object in the image of the functor. Using monads, a procedure that takes a value of type A to a value of type B, while possibly invoking some computational effect, can be modeled by a morphism from A to MB, where M is some monad which embodies the effect. For example, a common way to model nondeterministic computation is to use the powerset monad, which assigns to each set its set of subsets, so the output subset is the set of possible values the computation may yield.

To address the limitations of PCAs, this paper introduces Monadic Combinatory Algebras (MCAs), a novel generalization of PCAs designed to encapsulate a broader spectrum of computational effects by integrating monads into the combinatory algebra framework. Leveraging the monadic structure enables MCAs to internalize various side effects, such as non-deterministic computations, stateful computations, exceptions, and more. Hence, MCAs can go beyond partiality and seamlessly integrate a wide range of effects through the underlying monad. In doing so they provide a powerful framework for *internally* supporting effectful computations, which makes them particularly valuable in modern computational contexts. In fact, we show that MCAs can be instantiated to capture known effectful frameworks, such as non-determinism, stateful computation, continuations and parametric realizability.

We further provide a categorical characterization of MCAs within the context of Freyd categories, in the spirit of a similar characterization previously known of PCAs in Turing categories [6]. Specifically, we define a notion of a combinatory object in a Freyd category, and show that for monads in the category of sets, **Set**, the algebraic notion of an MCA precisely coincides with the categorical notion of a combinatory object in the Kleisli category $\mathbf{Set}_M$ of M over **Set**, which forms a Freyd category.

Finally, we explore the application of MCAs in the context of realizability theory, building upon the foundational role of PCAs in traditional realizability models [44]. Effectful realizability and more generally extensions of the Curry-Howad paradigm using effects to account for more advanced reasoning principles have been extensively studied over the last decades, with various approaches exploring its theoretical and practical implications [2, 3, 22, 31–34]. Among these, the role of monads has been pivotal in modeling and managing effects, offering a structured and algebraic perspective on effectful computation. Separately, the combinatorial approach to realizability has also garnered some new attention, emphasizing the benefits of working with combinators for algebraic purposes [18, 41–43]. Yet, regarding effects these

works were mostly focused on the particular case of Krivine realizability, i.e., computations manipulating continuations [41, 43]. In this work, we present the first integration of these perspectives, combining the algebraic insights of monads with the combinatorial methodology to offer a unified framework for realizability and monads.

Two standard realizability models stemming from PCAs are given by topoi and assemblies. From a categorical perspective, starting from a PCA one would define a tripos, and then use the tripos-to-topos construction to get a topos [20]. Properties of this realizability topos can be studied more easily directly into its subcategories of assemblies, which again can be defined over any PCA. Cohen *et al.* observed that effectfull realizability models could be defined in a uniform way using a structure called *evidenced frame*, factorizing the usual construction of a tripos from a PCA and making it compatible with effectful computational system [10]. Leveraging this, we here define evidenced frames over general MCAs, and even extend this approach to consider assemblies over any evidenced frame. This illustrates how traditional approaches to realizability based on PCAs easily generalize to MCAs. Moreover, the utility of the MCA framework is highlighted through several examples of known realizability models that arise naturally through the uniform MCA-based constructions.

Outline and Main contributions.

- Sec. 2 reviews the necessary background on PCAs and monads.
- Sec. 3 formally defines the novel structure of Monadic Combinatory Algebras, extending the foundational structure of PCAs to support a wide range of computational effects through underlying monads. The utility of the MCA framework is illustrated by modeling various computational effects (Sec. 3.2), and the categorical characterization of MCAs is established within Freyd Categories (Sec. 3.3).
- Sec. 4 shows how usual realizability semantics (namely triposes and assemblies) can be generalized to MCAs, via uniform constructions factoring through evidenced frames.
- Sec. 5 concludes with a summary of contributions and directions for future research.
- We supplement our development with an accompanying Rocq mechanization [9], hyperlinked with this paper via clickable icons.

2 Background

This section briefly overviews PCAs and monads, establishing key notations and conventions.

2.1 Partial Combinatory Algebras

Partial Combinatory Algebras (PCAs) provide a foundational model for the untyped λ-calculus and underlie many frameworks in computability, such as realizability theory. They are a generalization of combinatory algebras that allows for partial functions. That is, PCAs extend combinatory algebras by incorporating the computational effect of non-termination.

The definition of partial combinatory algebra is based on *partial applicative structures*.

▶ **Definition 1** (Partial Applicative Structure). *A partial applicative structure is a set $\mathbb{A}$ of "codes" equipped with a partial binary "application" operator on $\mathbb{A}$: $(-)\cdot(-) : \mathbb{A} \times \mathbb{A} \rightharpoonup \mathbb{A}$.*

We use $c_f \cdot c_a \downarrow c_r$ to denote c_r being the (successful) result of the application $c_f \cdot c_a$, and $c_f \cdot c_a \downarrow$ to denote that there is a code $c_r \in \mathbb{A}$ such that $c_f \cdot c_a \downarrow c_r$.

A PCA is then defined as a partial applicative structure that is "functionally complete", meaning there is a way to encode application expressions (such as $(c_1 \cdot (c_2 \cdot c_3)) \cdot (c_4 \cdot c_5)$)

with n free variables as individual codes accepting n arguments through applications. That is, we ensure that every formal expression involving variables, codes, and application, can be "internalized" in a similar vein to the abstraction done in λ-calculus. This ensures the necessary expressiveness for modeling computational systems like the λ-calculus.

To present the formal definition, we first formalize expressions e with numbered free variables $i \in \mathbb{N}$. To make the formalism easier to mechanize, we opt for lexical addresses (de Bruijn level notation), rather than nominal variables. Lexical addresses are numbers representing the index of a formal parameter within the body of a bound expression. For example, given a function $f(x_0, x_1, x_2)$, 2 is the lexical address of x_2 because it is the third parameter, so instead of writing x_2 within the body of f, one can use the number 2.

$$e ::= i \in \mathbb{N} \mid c \in \mathbb{A} \mid e \bullet e \qquad\qquad E_n(\mathbb{A}) ::= \{e \mid \text{all } i\text{s in } e \text{ are } < n\}$$

Term application $\bullet$ is left associative. Elements of $E_0(\mathbb{A})$ are called *closed* , while for every $n > 0$, elements of $E_n(\mathbb{A})$ are called *open*. Next, we define substitution $e\{c_a\}$ and evaluation to expressions $e \downarrow c_r$ as follows:

e	$e\{c_a\}$
0	c_a
$i+1$	i
c	c
$e_1 \bullet e_2$	$e_1\{c_a\} \bullet e_2\{c_a\}$

$$\frac{}{c \downarrow c} \qquad\qquad \frac{e_f \downarrow c_f \qquad e_a \downarrow c_a \qquad c_f \cdot c_a \downarrow c_r}{e_f \bullet e_a \downarrow c_r}$$

▶ **Definition 2** (Partial Combinatory Algebra)**.** *A* partial combinatory algebras (PCA) *is a partial applicative structure* $\mathbb{A}$ *such that for every expression* $e \in E_{n+1}(\mathbb{A})$ *there is a code* $\langle \lambda^n.e \rangle \in \mathbb{A}$*, satisfying the following laws:*

$$\langle \lambda^{n+1}.e \rangle \cdot c_a \downarrow \langle \lambda^n.e\{c_a\} \rangle \qquad\qquad \langle \lambda^0.e \rangle \cdot c_a \downarrow c_r \iff e\{c_a\} \downarrow c_r$$

The code $\langle \lambda^n.e \rangle$ is essentially the closure of the open expression e, embodying the λ-calculus term binding the $n+1$ free variables in e. Note that while we use this λ-construct to specify the result of a partial application, the above definition is equivalent (under certain assumptions) to the perhaps more traditional PCA definition [19].

2.2 Monads

Roughly speaking, monads are used to relate a computational effect with a specific endofunctor, say M, and consider effectful programs that return results of type A in context X as morphisms $f \in \mathbb{C}(X, MA)$. A *monad* M over category $\mathbb{C}$ is a functor $M : \mathbb{C} \to \mathbb{C}$, equipped with two natural transformations $\eta : \mathbf{1} \Rightarrow M$ and $\mu : MM \Rightarrow M$, satisfying equational laws, such that it forms a monoid object in the category of endofunctors over $\mathbb{C}$.

In the general case, interpreting computational effects in arbitrary categories requires a *strong monad* rather than just a monad [28]. However, here, for simplicity, we focus on a set theoretic settings, where every monad over the category of sets is a strong monad. Hence, we sometimes use a notation similar to the one from λ_c calculus [27], where $[a]$ stands for $\eta(a)$, often called 'return', and $\mathsf{let}\ x \Leftarrow m\ \mathsf{in}\ n$ stands for $\mu \circ M(\lambda x.n)(m)$, often called 'bind'.

Given a category $\mathbb{C}$ and a monad M on $\mathbb{C}$, the *Kleisli category* $\mathbb{C}_M$ has the same objects as $\mathbb{C}$, while its morphisms are Kleisli morphisms, i.e., $\mathbb{C}_M(A, B) := \mathbb{C}(A, MB)$. The identity morphism over an object A is $\eta_A \in \mathbb{C}(A, MA)$. Composition in $\mathbb{C}_M$ is defined for $f \in \mathbb{C}_M(B, C)$ and $g \in \mathbb{C}_M(A, B)$ by $\mu \circ Mf \circ g \in \mathbb{C}(A, MC)$.

3 Monadic Combinatory Algebras

This section presents the generalization of the notion of PCA to one that supports a wide range of computational effects. For this, we employ the standard structure for describing and reasoning about general effects: (strong) monads [28]. This categorical model has the benefit of preserving many useful properties of functions while allowing a wider variety of models. Thus, Sec. 3.1 defines Monadic Combinatory Algebra (MCAs), generalizing PCAs by encapsulating computational effects via monads, Sec. 3.2 demonstrates their versatility in embedding computational effects, and Sec. 3.3 provides their categorical characterization.

3.1 The Effectful Algebra

Just as PCAs describe the application as a *partial* operator, our generalized notion describes the application as an *effectful* operator, parameterized by some **Set** monad.

▶ **Definition 3** (Monadic Applicative Structure 🐘). *Given a* **Set** *monad* M*, a* Monadic Applicative Structure *(MAS) over* M *is a set of "codes"* $\mathbb{A}$ *with an application Kleisli function:* $(-)\cdot(-) : \mathbb{A} \times \mathbb{A} \to M\mathbb{A}$.

Terms with variables and substitution follow the PCA grammar. We think of $(-)\cdot(-)$ as a function that takes two codes and returns a 'computation' of a code. Intuitively, codes are either the encoding of programs, or encodings of inputs of programs. Using the same encoding for both allows us to discuss programs that take encodings of other programs as inputs. That is, viewing a code c_f as the encoding of a program, and a code c_a as the encoding of its input, $c_f \cdot c_a$ denotes the computation executed by running the program encoded by c_f on c_a as an input. So far, this is identical to the PCA settings. However, whereas the PAS application function can only produce a code (if anything at all), the MAS application function can produce any computational effect describable by a monad.

To go from the applicative structure to an algebra, we must employ some evaluation map that given a term, 'computes' the 'value' of the term. The key to the evaluation is the way in which it connects to the underlying monad [45]. Intuitively, the evaluation of a code simply returns the code, without exhibiting any effectful behavior. Application terms, on the other hand, evaluate using the monadic bind, depending on the specific strategy, and may exhibit the computational effect carried by the monad.

In principle, MCAs are orthogonal to the specifics of the evaluation strategy. That is, we can define a notion of an MCA based on various evaluation strategies. In this paper, to conform with traditional PCA, we opt to commit to the common Call-by-Value (CbV) evaluation strategy, codified in the following evaluation model.

▶ **Definition 4** (CbV Evaluation 🐘). *Evaluation* ν *is defined by induction on* $E_0(\mathbb{A})$ *as follows:*

$$\nu(c) := [c] \qquad \nu(e_f \bullet e_a) := \mathsf{let}\ c_f \Leftarrow \nu(e_f)\ \mathsf{in}\ \mathsf{let}\ c_a \Leftarrow \nu(e_a)\ \mathsf{in}\ c_f \cdot c_a$$

Note that evaluation is reflected by an *equality* in $\mathbb{A}$. However, this could be generalized to an ordered setting following [18, 41].

Next, we turn to the definition of monadic combinatory algebras. To ensure they can support general computation, they must be at least as computationally powerful as the λ-calculus. This is done through a similar mechanism to how PCAs are defined. But, unlike PCAs, which permit only purely functional behavior (up to non-termination), monadic combinatory algebras accommodate any computational effect representable via monads.

▶ **Definition 5** (Monadic Combinatory Algebra 🦕). *A Monadic Combinatory Algebra (MCA) is a monadic applicative structure $\mathbb{A}$ such that for every expression $e \in E_{n+1}(\mathbb{A})$ there is a code $\langle \lambda^n.e \rangle \in \mathbb{A}$ satisfying the following laws:*

$$\begin{array}{lll} \forall n \in \mathbb{N}. \forall e \in E_{n+2}(\mathbb{A}). \forall c \in \mathbb{A}. & \langle \lambda^{n+1}.e \rangle \cdot c & = \eta(\langle \lambda^n.e\{c\} \rangle) \\ \forall e \in E_1(\mathbb{A}). \forall c \in \mathbb{A}. & \langle \lambda^0.e \rangle \cdot c & = \nu(e\{c\}) \end{array}$$

The *abstraction* assignment of codes to expressions turns open terms into codes that internalize their evaluation after substitution using the monad's return and bind. Principally, the MCA laws state that it "delays" the evaluation of an open term it encloses until all free variables in it are substituted by codes. If there is more than one free variable in the term, using η means the application of the abstracted term to a code only substitutes the outermost parameter with the argument. However, if the term has exactly one free variable, applying it to the argument obtains a closed term, which is then immediately evaluated.

Terms that only involve codes given by the abstraction operator alone are called *pure terms*. The MCA definition closely resembles that of a PCA; in fact, when evaluating pure terms, the behavior remains identical to a PCA, exhibiting no computational effects beyond non-termination. Thus, computational effects arise only through additional codes not definable via abstraction. For example, when describing non-deterministic computation, using the powerset monad, the evaluation of pure terms will always result in a set with at most one value, while using the monad allows $\mathbb{A}$ to have codes which, when applied, yield a set with multiple values (see Sec. 3.2). However, the monad still plays an important role when evaluating pure terms, as it allows interpreting a non-terminating evaluation as special elements within the set of computations, denoting the absence of a value. Concretely, a PCA is obtained from an MCA by instantiating the monad with the sub-singleton monad, i.e. MA is the set of subsets of A in which all elements are equal.

$$MA = \{S \subseteq A \mid \forall x_1, x_2 \in S. x_1 = x_2\} \qquad \eta_A(x) = \{x\} \qquad \mu_A(m) = \bigcup_{X \in m} X$$

▶ **Proposition 6.** *PCA is a special case of an MCA.*

Def. 5 is equivalent to the (perhaps more familiar) formalization of combinatory completeness via the s and k combinators [44]. That is, we could have alternatively, defined an MCA as a MAS with s and k combinators, because s and k are simply encodings of particular expressions that are sufficient to ensure that all expressions can be encoded, and, in the converse direction, the above MCA laws essentially ensure their existence. Note, however, that the characterizing axioms for these combinators in the discourse are tailored to a non-effectful behavior of the calculus. Standardly, the axioms of s and k require all their partial applications to be defined. Usually written as $\mathsf{k} \cdot c_1 \downarrow$, $\mathsf{s} \cdot c_1 \downarrow$, and $\mathsf{s} \cdot c_1 \cdot c_2 \downarrow$ for any c_1 and c_2. In the context of MCAs, however, it is not sufficient for the partial applications to be defined, but they also have to be pure, not triggering any effects beyond the evaluation of their value. This constraint ensures s and k do exactly the same thing they do in PCAs, and nothing else, regardless of the underlying monad. For monads, purity is represented with the monad unit. Hence, the monadic version of the axioms requires the partial applications of s and k to return the application of the monad unit over particular codes.

▶ **Proposition 7** (🦕). *An MCA is equivalent to a MAS with codes $e_{\mathsf{S}}, e_{\mathsf{K}}, e_{\mathsf{S}(c_1)}, e_{\mathsf{S}(c_1,c_2)}$, and $e_{\mathsf{K}(c_1)}$ for any two codes c_1, c_2, satisfying the following axioms:*

$$\begin{array}{llll} e_{\mathsf{S}} \cdot c_1 = [e_{\mathsf{S}(c_1)}] & e_{\mathsf{S}} \cdot c_1 = [e_{\mathsf{S}(c_1)}] & e_{\mathsf{S}(c_1,c_2)} \cdot c_3 = \nu((c_1 \bullet c_3) \bullet (c_2 \bullet c_3)) \\ e_{\mathsf{K}} \cdot c_1 = [e_{\mathsf{K}(c_1)}] & e_{\mathsf{K}(c_1)} \cdot c_2 = [c_1] & \end{array}$$

3.2 MCA Instances

As demonstrated, PCAs are a special case of MCAs. However, the generalized structure also encompasses many common effectful structures proposed in the literature. This section illustrates how those can be derived from an MCA by instantiating the underlying monad.

3.2.1 Relational Combinatory Algebra

Relational Combinatory Algebras (RCAs) were defined in [8, 10] to account for (demonic) non-determinism. Concretely, they were used to show that while all realizability models stemming from PCAs model the principle of Countable Choice, realizability models based on RCAs can, in fact, model the negation of the principle.

RCAs correspond to an MCA with M being the *powerset* monad, as in Fig. 1. The powerset monad captures non-deterministic computations by considering the subset of possible results. When $\mathbb{A}$ is an RCA, it may have codes such as e_{flip}, which nondeterministically returns either $\langle \lambda^1.0 \rangle$ or $\langle \lambda^1.1 \rangle$ whenever it is applied. That behavior is defined by representing the set of possible return values when describing the application of e_{flip} : $e_{\mathsf{flip}} \cdot c = \{\langle \lambda^1.0 \rangle, \langle \lambda^1.1 \rangle\}$.

3.2.2 Stateful Combinatory Algebra

Stateful Combinatory Algebras (SCAs) were defined in [8, 10] as a stateful extension of RCAs. SCAs were used to memoize non-deterministic computations and recover a realizer of Countable Choice even in the presence of nondeterminism.

SCAs correspond to MCAs where M is the *powerset state* monad, as given in Fig. 1. This allows taking a code in a given state and returning a set of all possible pairs of results in new states. In [8,10], a variant of the *increasing* state monad was used, which is a submonad of the powerset state monad: $MA = \{m : \Sigma \to \mathcal{P}(\Sigma \times A) \mid \forall \sigma_0 \in \Sigma. \forall (\sigma_1, x) \in m(\sigma_0). \sigma_0 \leq \sigma_1\}$.

When $\mathbb{A}$ is an SCA, it may have codes such as e_{get} and e_{inc}, implementing a counter. For simplicity, we take states to be natural numbers. When e_{get} is applied to a code c, it ignores c and returns the Church numeral representing the current state of the counter, leaving the state intact. When e_{inc} is applied to a code c, it increments the counter and returns c.

$$e_{\mathsf{get}} \cdot c = \lambda n. \{(\overline{n}, n)\} \qquad e_{\mathsf{inc}} \cdot c = \lambda n. \{(c, n+1)\}$$

3.2.3 CPS Combinatory Algebra

The double-negation translation [16], relating classical logic with intuitionistic logic, points to a connection between classical logic and the continuation monad. This connection has been extensively studied, particularly in the context of classical realizability [22]. However, most work on classical realizability is based on Krivine abstract machines rather than combinatory algebras, which is an entirely different model of computation. By utilizing MCAs with the continuation monad, we can align classical realizability with a computational model akin to intuitionistic realizability.

Here, we focus on Continuation-Passing-Style (CPS), which is a style of programming in which control is explicitly passed through continuation functions. Thus, instead of returning results directly, functions in CPS receive an extra argument: a continuation function that specifies what to do next with the result. A *CPS Combinatory Algebra* (CPSCA) is an MCA where the underlying monad is the CPS monad, as described in Fig. 1.[1] The CPS

[1] In general, classical realizability is not constructed via combinatory algebras, but there are similar

Comb. Alg.	**Monad** MA	**return** $\eta_A(x)$	**bind** $\mu_A(m)$
Partial	$\{X \subseteq A \mid \forall x, y \in X. x = y\}$	$\{x\}$	$\bigcup_{X \in m} X$
Relational	$\mathcal{P}(A)$	$\{x\}$	$\bigcup_{X \in m} X$
Stateful	$\Sigma \to \mathcal{P}(\Sigma \times A)$	$\lambda\sigma.\{(\sigma, x)\}$	$\lambda\sigma. \bigcup_{(\sigma', f) \in m(\sigma)} f(\sigma)$
CPS	$(A \to R) \to R$	$\lambda k. k(x)$	$\lambda k. m(\lambda g. g(k))$
Parameterized	$\mathbf{P} \to \{X \subseteq A \mid \forall x, y \in X. x = y\}$	$\lambda p. \{x\}$	$\lambda p. \bigcup \{g(p) \mid g \in m(p)\}$

Figure 1 MCA Instances

monad allows for composable and reusable continuations, i.e., it models computations with direct access to the call stack, enabling non-trivial control flow manipulation. In the above definition, R can be any set, representing the ultimate results of the whole computation.

When $\mathbb{A}$ is a CPSCA, it may have codes such as e_{cc} and e_{K_u}, which save and replace the current continuation:

$$(e_{\mathsf{cc}} \cdot c_a)(u) = (c_a \cdot e_{\mathsf{K}_u})(u) \qquad (e_{\mathsf{K}_u} \cdot c_a)(u') = u(c_a)$$

Using CPSCAs, the definitions of evaluation and application can be seen as a CPS form of an evaluator for PCAs. By defunctionalization, this leads directly to an *eval/apply* abstract stack machine, providing operational semantics for PCAs [11]. The machine has a stack, which can hold either codes c tagged with $\boldsymbol{v}(c)$, or terms e tagged with $\boldsymbol{t}(e)$. An empty stack is marked as $\emptyset$, while a nonempty stack is marked with $x : \pi$, where x is the top of the stack and π is the rest. The machine has three states:

1. *Eval* state, marked as $e \triangleright \pi$, in which the machine takes a closed term e, and a stack π.
2. *Apply* state, marked as $c \blacktriangleleft \pi$, in which the machine takes a value c, and a stack π.
3. *Final* state, marked as simply a code c, of the final value.

The semantics of the machine is defined via a one-step transition relation $\hookrightarrow$ between states:

$$\begin{array}{rcl|rcl} e_f \cdot e_a \triangleright \pi & \hookrightarrow & e_f \triangleright \boldsymbol{t}(e_a) : \pi & c_f \blacktriangleleft \boldsymbol{t}(e_a) : \pi & \hookrightarrow & e_a \triangleright \boldsymbol{v}(c_f) : \pi \\ c \triangleright \pi & \hookrightarrow & c \blacktriangleleft \pi & c_a \blacktriangleleft \boldsymbol{v}(\langle \lambda^0.e \rangle) : \pi & \hookrightarrow & e\{c_a\} \triangleright \pi \\ c \blacktriangleleft \emptyset & \hookrightarrow & c & c_a \blacktriangleleft \boldsymbol{v}(\langle \lambda^{n+1}.e \rangle) : \pi & \hookrightarrow & \langle \lambda^n.e\{c_a\} \rangle \blacktriangleleft \pi \end{array}$$

3.2.4 Parameterized Combinatory Algebra

The notion of *Parameterized Combinatory Algebras* (ParCAs), introduced by Bauer and Hanson in [1], is based on a notion of computations that has access to external oracles. Using parameterized combinatory algebras the authors constructed a model in which the set of Dedekind real numbers is countable.

A ParCA is precisely an MCA based on the *subsingleton reader* monad. That is, for a set $\mathbf{P}$ of parameters, the parameterized monad is defined in Fig. 1. A function into the subsingleton reader monad represents a partial computation, which, in addition to its input, has access to some external parameters in a set $\mathbf{P}$. The uniformity of the MCA framework here is highlighted by the fact that the comprehensive algebraic axiomatization of ParCA in [1] is naturally derivable from the MCA representation.

Computations with external oracles can be modelled by simply allowing the MCA to have access to external parameters. When $\mathbb{A}$ is a ParCA, where the parameter is taken from

structures, e.g. [41] which uses ordered combinatory algebras, or [23] which uses abstract machines.

predicates over $\mathbb{A}$, that is $\mathbf{P} = \mathbb{A} \to \{0,1\}$, it may have a code such as e_{search}. When e_{search} is applied to another code c, it takes a predicate p as parameter, and returns either $\langle \lambda^1.0 \rangle$ or $\langle \lambda^1.1 \rangle$ according to whether c satisfies p:

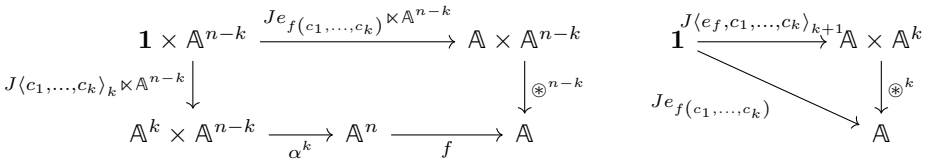
$$(e_{\mathsf{search}} \cdot c) = \lambda p.\mathtt{if}\ p\,(c) = 0\ \mathtt{then}\ \{\langle \lambda^1.0 \rangle\}\ \mathtt{else}\ \{\langle \lambda^1.1 \rangle\}$$

3.3 Categorical Characterization of MCAs

PCAs have been given a categorical representation in [6], by establishing their connection to Turing categories [6, 25]. Concretely, it was shown that PCAs are PCA-objects in the category of sets, which are essentially the categorical counterpart of the notion of combinatory completeness for PCAs. To obtain a similar categorical characterization for the generalized notion of MCAs which is based on arbitrary monads, we here work within Freyd categories [24, 39] which are an extension of the categorical framework designed specifically to model computational effects. Importantly, they abstract the structure of the Kleisli category of a monad by employing one category for values and another one for computations. Due to space limitations, we here provide the key results, leaving full details to the appendix.

▶ **Definition 8** (Applicative Object). *For a Freyd category $\langle \mathbb{C}, \mathbb{K}, J \rangle$, an* applicative object *is an object $\mathbb{A} \in \mathbf{Obj}\,(\mathbb{C})$ equipped with an application morphism $\circledast \in \mathbb{K}\,(\mathbb{A} \times \mathbb{A}, \mathbb{A})$.*

▶ **Definition 9** (Computable Morphism). *Given an applicative object $\mathbb{A}$ in a Freyd category $\langle \mathbb{C}, \mathbb{K}, J \rangle$ and an $0 < n \in \mathbb{N}$, we say that a $\mathbb{K}$ morphism $f \in \mathbb{K}\,(\mathbb{A}^n, \mathbb{A})$ is $\mathbb{A}$-computable when, for all $k \in \{0, \dots, n-1\}$, and all $c_1, \dots, c_k \in \mathbb{C}\,(\mathbf{1}, \mathbb{A})$, there is a code $e_{f(c_1,\dots,c_k)} \in \mathbb{C}\,(\mathbf{1}, \mathbb{A})$ such that the following diagrams commute in $\mathbb{K}$ (where e_f is $e_{f(c_1,\dots,c_k)}$ for $k = 0$).*



Our definition of a combinatory object relies on the notion of $\mathbb{A}$-monomials, generalizing [25]. An $\mathbb{A}$-monomial is a morphism in $\mathbb{K}\,(\mathbb{A}^n, \mathbb{A})$, for some $n \in \mathbb{N}$, defined using only projections, morphisms in $\mathbb{C}\,(\mathbf{1}, \mathbb{A})$, and application $\circledast$, used in an applicative order. For $n > 0$ the $\mathbb{A}$-monomial is called positive.

▶ **Definition 10** (Combinatory object). *An applicative object $\mathbb{A}$ is called a* combinatory object *when all positive $\mathbb{A}$-monomials are $\mathbb{A}$-computable.*

▶ **Theorem 11.** *Let M be a* **Set** *monad. $\mathbb{A}$ is an MCA over M if and only if it is a combinatory object in $\mathbf{Set}_M$.*

Furthermore, the definition of a PCA-object in a cartesian restriction category [6], is the exact counterpart of a combinatory object in a Freyd category, obtained by replacing morphisms of a Freyd category with their counterparts in a cartesian restriction category.

4 MCA-induced Realizability Models

Since PCAs underpin traditional realizability models, this section explores the application of MCAs in the broader context of realizability theory. By incorporating computational

effects, MCAs broaden realizability models, enabling a more extensive semantic framework for diverse logic systems based on computation.

Evidenced frames propose a unifying approach to effectful realizability [10]. An evidenced frame is a structure that abstracts the core components of realizability models by focusing solely on the relationship between propositions and their evidence, while omitting the computational specifics of individual models. The evidenced frame abstraction is complete in that any realizability tripos (i.e. a model of higher-order dependent predicate logic) can be viewed as an evidenced frame, and there is a uniform construction generating a corresponding realizability tripos from an evidenced frame. Besides, the usual construction of a tripos from a PCA smoothly factorizes through the definition of an evidenced frame.

To provide semantic realizability models from an MCA, we build on this construction and demonstrate how an evidenced frame can be derived from it. This establishes a clear pathway from MCAs to realizability triposes, and in turn, through the tripos-to-topos construction [36], to a realizability topos, which is a model of (extensional, impredicative) dependent type theory (and set theory). It further factors an alternative construction of assemblies which, in the case of PCAs, have been broadly studied in relation to the realizability topos.

The following development uses preordered sets, and in particular, complete Heyting prealgebras, for the interpretation of logic. A preordered set $(\Omega, \leq)$ is a set Ω equipped with a reflexive and transitive "inequality" relation $\leq$ over Ω. A Heyting prealgebra extends a preordered set by additional algebraic operations, namely, meet, join and implication. In the context of semantics of logic, a preordered set Ω is used as a set of "truth values", and formulas in the logic are interpreted as elements of Ω. That is, for ϕ a formula, its interpretation $[\![\phi]\!]$ is an element of Ω. The preorder relation of a Heyting prealgebra corresponds to the logical entailment $\vdash$ between formulas, while the algebraic counterpart of universal (resp. existential) quantifications is provided by meets $\sqcap$ (resp. joins $\sqcup$). The bottom and top elements of Ω are denoted $\mathbf{0}$ and $\mathbf{1}$ (resp.). This will come in handy in order to give an algebraic structure to the predicates of our realizability models.

4.1 Realizability Triposes via Evidenced Frames

First, we recall the definition of an evidenced frame.

▶ **Definition 12** (Evidenced Frame [10]). *An* evidenced frame *is a triple* $\mathcal{EF} = \left(\Phi, E, \cdot \overset{\cdot}{\to} \cdot\right)$, *where* Φ *is a set of propositions,* E *is a set of evidence, and* $\phi_1 \overset{e}{\to} \phi_2$ *is a ternary evidence relation on* $\Phi \times E \times \Phi$*, along with the structure captured in Fig. 2.*

The evidenced frame setup mirrors computational processes, where evidence can be thought of as programs or computational artifacts that demonstrate logical relationships. In fact, evidenced frames supplement complete Heyting prealgebras, which are standard models of intuitionistic logic, by adding computational evidence for the validity of the preorder relation. Hence, while in a prealgebra, $\phi_1 \leq \phi_2$ denotes that the pair (ϕ_1, ϕ_2) satisfies the preorder relation, in an evidenced frame the corresponding notion is $\phi_1 \overset{e}{\to} \phi_2$, meaning the triple (ϕ_1, e, ϕ_2) satisfies the evidenced relation. All the axioms of a complete Heyting prealgebra, described in terms of the preorder relation $\leq$, appear in an evidenced frame in an enhanced form, where $\leq$ is replaced with the evidence relation $\cdot \overset{\cdot}{\to} \cdot$, and each axiom requires the existence of some "evidence" (appearing as e in $\phi_1 \overset{e}{\to} \phi_2$) which uniformly witnesses the validity of the axiom. Evidenced frames are flexible in that they do not assume a specific equational theory which allows them to capture a broader spectrum of computational effects and unify them by focusing on uniform evidence.

	Logical Const.	Program Const.	Evidence Relation
Reflexivity		$e_{\mathtt{id}} \in E$	$\phi \xrightarrow{e_{\mathtt{id}}} \phi$
Transitivity		$; \in E \times E \to E$	$\phi_1 \xrightarrow{e_1} \phi_2 \wedge \phi_2 \xrightarrow{e_2} \phi_3 \Rightarrow \phi_1 \xrightarrow{e_1 \,;\, e_2} \phi_3$
Top	$\top \in \Phi$	$e_\top \in E$	$\phi \xrightarrow{e_\top} \top$
Conjunction	$\wedge \in \Phi \times \Phi \to \Phi$	$\langle\!\langle \cdot,\cdot \rangle\!\rangle \in E \times E \to E$ $e_{\mathtt{fst}}, e_{\mathtt{snd}} \in E$	$\phi \xrightarrow{e_1} \phi_1 \wedge \phi \xrightarrow{e_2} \phi_2 \Rightarrow \phi \xrightarrow{\langle\!\langle e_1, e_2 \rangle\!\rangle} \phi_1 \wedge \phi_2$ $\phi_1 \wedge \phi_2 \xrightarrow{e_{\mathtt{fst}}} \phi_1 \quad , \quad \phi_1 \wedge \phi_2 \xrightarrow{e_{\mathtt{snd}}} \phi_2$
Universal Implication	$\supset \in \Phi \times \mathcal{P}(\Phi) \to \Phi$	$\lambda \in E \to E$ $e_{\mathtt{eval}} \in E$	$(\forall \phi \in \vec{\phi}.\ \phi_1 \wedge \phi_2 \xrightarrow{e} \phi) \Rightarrow \phi_1 \xrightarrow{\lambda e} \phi_2 \supset \vec{\phi}$ $\forall \phi \in \vec{\phi}.\ (\phi_1 \supset \vec{\phi}) \wedge \phi_1 \xrightarrow{e_{\mathtt{eval}}} \phi$

Figure 2 Evidenced Frame constructs, where $\vec{\phi} \in \mathcal{P}(\Phi)$ and the evidence relations are universally quantified.

4.1.1 M-Modalities

As shown in [10], in the standard case of a PCA, $\mathbb{A}$, the corresponding evidenced frame is that in which E is the set of codes $\mathbb{A}$, Φ is its set of subsets $\mathcal{P}(\mathbb{A})$ and for every $\phi_1, \phi_2 \in \mathcal{P}(\mathbb{A})$ and $c_f \in \mathbb{A}$, $\phi_1 \xrightarrow{c_f} \phi_2$ stands for: $\forall c_a \in \mathbb{A}. c_a \in \phi_1 \Rightarrow \exists c_r.\, c_f \cdot c_a \downarrow c_r \wedge c_r \in \phi_2$. Our goal here is to generalize the embedding of combinatory algebras into evidenced frames and provide a uniform construction of an evidenced frame from any MCA. However, the MCA abstraction poses a challenge. For PCAs, the result of the computation $c_f \cdot c_a$ is given by the reduction predicate $c_f \cdot c_a \downarrow c_r$, and then related to a subset ϕ_2 through the membership predicate $c_r \in \phi_2$. However, in an MCA, the result $c_f \cdot c_a$ is in $M(\mathbb{A})$, so it appears within the context of a more abstract notion of computation, and some device is needed to be able to pick the result from the computational context and relate it to a subset, or more generally, to a predicate. In fact, traditional realizability manipulates codes, not computations, and so our device needs to be able to extend predicates defined on values to predicates on computations, which then can be used again in the realizability setting. As we discuss below, we will consider predicates on a set X as defined by functions from X to some Heyting prealgebras Ω, hence our device will essentially extend such a function to a function in $M(X) \to \Omega$.

The device appropriate for this task is called an M-modality, which intuitively describes a post-condition over the result of a (possibly effectful) computation. A variant of the notion of an M-modality first appeared in [28] and then further elaborated in [35].[2]

▶ **Definition 13** (M-modality 🐓). *Let M be a* **Set** *monad, $\mathbb{A}$ an MCA over M, and $(\Omega, \leq)$ a complete Heyting prealgebra. An M-modality over Ω is a natural transformation:*

$$\Diamond_X : M(X) \to (X \to \Omega) \to \Omega \qquad \textit{and we note } \langle\!\langle x \leftarrow m \rangle\!\rangle \phi(x) := \Diamond(m)(\phi)$$

satisfying for all A, B, $\phi_i : A \to \Omega$, $f : A \to M(B)$, $a \in A$ and $m \in M(A)$ the following:

After-Return. $\quad \phi(a) \leq \langle\!\langle x \leftarrow [a] \rangle\!\rangle \phi(x)$

After-Bind $\quad \langle\!\langle x \leftarrow m \rangle\!\rangle \langle\!\langle y \leftarrow f(x) \rangle\!\rangle \phi(y) \leq \langle\!\langle y \leftarrow \mathsf{let}\ x \Leftarrow m\ \mathsf{in}\ f(x) \rangle\!\rangle \phi(y)$

Internal Monotonicity. $\quad \bigsqcap_c (\phi_1(c) \sqsupset \phi_2(c)) \leq \langle\!\langle x \leftarrow m \rangle\!\rangle \phi_1(x) \sqsupset \langle\!\langle x \leftarrow m \rangle\!\rangle \phi_2(x)$

where we apply the standard precedence of quantifiers, for example $\langle\!\langle x \leftarrow m \rangle\!\rangle \phi_1(x) \sqsupset \phi_2(x)$ is to be read as $\left(\langle\!\langle x \leftarrow m \rangle\!\rangle \phi_1(x)\right) \sqsupset \phi_2(x)$.

[2] Def. 13 can equivalently be formulated as an oplax algebra $\omega : M\Omega \to \Omega$, similar to the T-modal operator in [28]. However, this will require the definition of each particular modality to operate on truth values in Ω, rather than to relate value predicates with computations, which seems easier in practice.

Intuitively, Ω is a set of truth values, so a predicate ϕ over a set X is a function $\phi : X \to \Omega$, and to denote that ϕ applies to some $x \in X$, we write $\phi(x)$ as is standard. As for the modality, we read $\langle\!\langle x \leftarrow m \rangle\!\rangle \phi(x)$ as saying that after the computation m yields a value x (in case it does), then $\phi(x)$ holds. To obtain a sound logical framework, the properties of an M-modality ensure it is well-behaved with respect to the computational operators of the monad and the logical operators of the complete Heyting prealgebra.

The use of Heyting prealgebras, rather than subsets, allows us to generalize the standard notion of a subset of codes to more complex subset-like structures, in particular, ones that account for the computational effects and enforce invariants over the computational behavior. For example, in the case of stateful nondeterministic computation, as in Sec. 3.2.2, it is useful to consider subsets of pairs of codes and states, $\mathcal{P}(\Sigma \times \mathbb{A})$, and to obtain a well-behaved logic we must restrict attention to "future-stable" predicates, i.e. predicates which, for every nondeterministic stateful computation, if they hold before the change of state, they keep holding for every possible mutation of the state as well. In terms of subsets, it means the set of states has to be preordered, and instead of taking $\mathcal{P}(\Sigma \times \mathbb{A})$, we take $\mathcal{U}(\Sigma)^{\mathbb{A}}$, where $\mathcal{U}(\Sigma)$ is the set of all upper subsets of Σ. This set has the structure of a complete Heyting algebra (and thus, a complete Heyting prealgebra) as a topological space, given by the Alexandrov topology. However, the Heyting prealgebra structure of $\mathcal{P}(\Sigma \times \mathbb{A})$ alone is not enough for constructing an evidenced frame over SCAs. For this, we need to additionally require the modality preserves implication, which is not the case in the standard angelic and demonic interpretations of nondeterminism, but is guaranteed by internal monotonicity.

The internal monotonicity of the M-modality is stronger than the perhaps more well-known "order-preserving" property of the T-modality in [35]. Syntactically, internal monotonicity ensures that the modality preserves implication. Semantically, it ensures that any property that holds before a computation keeps holding afterwards.

To make sure the induced semantics are meaningful, we must verify that evidence for entailment does not exist for every pair of predicates. For example, in the case of PCAs, one can define the modality $\langle\!\langle x \leftarrow m \rangle\!\rangle \phi(x) := \bigsqcap_{x \in m} \phi(x)$ corresponding to partial correctness. However, with this modality, any code that yields no value when applied to any argument, such as $c_{\circlearrowleft} := \langle \lambda^0. \langle \lambda^0.0 \bullet 0 \rangle \bullet \langle \lambda^0.0 \bullet 0 \rangle \rangle$, can be used as evidence for the entailment of any pair of predicates. In particular, consider the predicates $\top = \lambda x.\mathbf{1}$ and $\bot = \lambda x.\mathbf{0}$, then $\top \overset{c_{\circlearrowleft}}{\to} \bot$ would mean $1 \leq \bigsqcap_{c \in \nu(c_{\circlearrowleft})} .0$. Since $\nu(c_{\circlearrowleft}) = \emptyset$, this statement is vacuously true, and thus we can consistently model an inconsistent theory.

To eliminate this option, we have to make sure the modality is selective enough to prevent absurd entailments from being evidenced. To that end, we employ a similar technique to the one mentioned in [10], with a generalized notion of a separator for our setting. As we shall see in Thm. 18, separators will indeed ensure the consistency of the induced evidenced frame.

▶ **Definition 14** (Separator 🦫). *Given an MCA $\mathbb{A}$, a Heyting prealgebra Ω and an M-modality $\Diamond$ over them, a separator for $\Diamond$ (or a $\Diamond$-separator) is a combinatory complete subset $\mathcal{S}$ of $\mathbb{A}$, such that, for every $c_f, c_a \in \mathcal{S}$, the following "progress" property holds: $\langle\!\langle r \leftarrow c_f \cdot c_a \rangle\!\rangle \mathbf{0} \leq \mathbf{0}$.*

One common separator is the one that consists of all codes (when progress holds for all of them). However, since our framework supports arbitrary forms of effectful computations, at times it will be necessary to exclude some codes from the separator. As a simple example, consider a set of codes that contains fail, defined such that $\mathsf{fail} \cdot c = \emptyset$ for every $c \in \mathbb{A}$. While we want to allow codes such as fail to be used to define and realize propositions, for consistency, fail cannot serve as valid evidence for entailment. A more subtle example of a separator for CPS continuations is given in Ex. 22.

In summary, defining realizability semantics from an MCA requires extra structure, captured by the following notion of a monadic core.

▶ **Definition 15** (Monadic Core). *A* monadic core *is a tuple* $\mathcal{MC}_M := (\mathbb{A}, \Omega, \Diamond, \mathcal{S})$*, where* $\mathbb{A}$ *is an MCA over a* **Set** *monad* M*,* $\Diamond$ *is an* M*-modality over* $\mathbb{A}$ *and a complete Heyting prealgebra* $(\Omega, \leq)$*, and* $\mathcal{S}$ *is a separator over them.*

4.1.2 From Monadic Cores to Evidenced Frames

The next theorem demonstrates how one can construct evidenced frames from a monadic core, i.e., an MCA and an associated M-modality. The propositions are taken to be functions from the MCA to the complete Heyting prealgebra underlying the M-modality. As explained, this is so that propositions are given an algebraic structure generalizing their usual definition as sets of codes. Evidence are elements of the separator (rather than arbitrary codes), and conceptually the evidence relation holds when the separator maps realizers for the input proposition to computations that, after they terminate, yield realizers for the output proposition.

▶ **Theorem 16** (Evidenced Frame over Monadic Core 🐓). *Let* $\mathcal{MC}_M = (\mathbb{A}, \Omega, \Diamond, \mathcal{S})$ *be a monadic core. The triple* $\left(\Omega^{\mathbb{A}}, \mathcal{S}, \cdot \dot{\to} \cdot\right)$ *forms an evidenced frame, where*

$$\phi_1 \xrightarrow{e} \phi_2 := \forall c \in \mathbb{A}. \phi_1(c) \leq \left\langle\!\!\left\langle r \leftarrow e \cdot c \right\rangle\!\!\right\rangle_{\Diamond} \phi_2(r).$$

Proof Sketch. We define the logical and program constructs, while the proofs that they satisfy the required properties can be found in [9]. Let $e_{\mathtt{p}_1} := \langle \lambda^1.0 \rangle$ and $e_{\mathtt{p}_2} := \langle \lambda^1.1 \rangle$.

Reflexivity: Take $e_{\mathtt{id}} := \langle \lambda^0.0 \rangle$.
Transitivity: Take $e_1 ; e_2 := \langle \lambda^0. e_2 \bullet (e_1 \bullet 0) \rangle$.
Top: Take $\top := \lambda e.\mathbf{1}$ and $e_\top := e_{\mathtt{id}}$.
Conjunction: Take $(\phi_1 \wedge \phi_2)(e) := \left\langle\!\!\left\langle c_1 \leftarrow e \cdot e_{\mathtt{p}_1} \right\rangle\!\!\right\rangle_{\Diamond} \phi_1(c_1) \sqcap \left\langle\!\!\left\langle c_2 \leftarrow e \cdot e_{\mathtt{p}_2} \right\rangle\!\!\right\rangle_{\Diamond} \phi_2(c_2)$, and $\langle\!\langle e_1, e_2 \rangle\!\rangle := \left\langle \lambda^1.1 \bullet (e_1 \bullet 0) \bullet (e_2 \bullet 0) \right\rangle, e_{\mathtt{fst}} := \left\langle \lambda^0.0 \bullet e_{\mathtt{p}_1} \right\rangle, e_{\mathtt{snd}} := \left\langle \lambda^0.0 \bullet e_{\mathtt{p}_2} \right\rangle$.
Universal Implication: Take $\phi \supset \vec{\phi}(e) := \bigsqcap_{\phi \in \vec{\phi}} \bigsqcap_{c \in \mathbb{A}} \left(\phi(c) \sqsupset \left\langle\!\!\left\langle r \leftarrow e \cdot c \right\rangle\!\!\right\rangle_{\Diamond} \phi(r)\right)$, and $\lambda e := \left\langle \lambda^1. e \bullet \left(\left\langle \lambda^2.2 \bullet 0 \bullet 1 \right\rangle \bullet 0 \bullet 1\right) \right\rangle, e_{\mathtt{eval}} := \left\langle \lambda^0. e_{\mathtt{id}} \bullet (0 \bullet e_{\mathtt{p}_1}) \bullet (0 \bullet e_{\mathtt{p}_2}) \right\rangle$. ◀

The above theorem, together with the UFam construction [10, Def. V.4] that constructs a realizability tripos from an evidenced frame, thus provides the following realizability semantics for MCAs. As for the standard PCA-based tripos, a set I is mapped to a family of propositions indexed by I, which is given a structure of Heyting prealgebra by considering the existence of a uniform realizer (i.e. compatible with any $i \in I$) to witness the preordering relation. As in the evidenced frame, the M-modality is used to handle computations instead of only values. Morphisms are simply mapped to reindexing functions along them.

▶ **Corollary 17.** *Let* $\mathcal{MC}_M = (\mathbb{A}, \Omega, \Diamond, \mathcal{S})$ *be a monadic core. Then the following functor* $\mathcal{T}$ *from* $\mathbf{Set}^{\mathrm{op}}$ *to the category of Heyting prealgebras is a tripos.*

$$\begin{array}{rcll} \mathcal{T}(I) & := & \left((\Omega^{\mathbb{A}})^I, \leq_I\right) & \mathcal{T}(f)(\varphi) := j \mapsto \varphi(f(j)) \\ \varphi \leq_I \psi & := & \exists e \in \mathcal{S}. \forall i \in I. \forall c \in \mathbb{A}. \varphi(i)(c) \leq \left\langle\!\!\left\langle r \leftarrow e \cdot c \right\rangle\!\!\right\rangle_{\Diamond} \psi(i)(r) & \end{array}$$

We can now verify that the use of a separator indeed ensures the consistency of the induced evidenced frame, as shown in the following theorem.

▶ **Theorem 18** (🐓). *Let* $\mathcal{MC}_M = (\mathbb{A}, \Omega, \Diamond, \mathcal{S})$ *be a monadic core. Then the induced evidenced frame has an evidence* $e \in \mathbb{A}$ *such that* $\top \xrightarrow{e} \bot$ *iff* $\mathbf{1} \leq \mathbf{0}$.

With that in mind, an M-modality over a non-trivial Ω is called *consistent* when it has a separator. For example, $\llparenthesis x \leftarrow m \rrparenthesis \phi(x) := \bigsqcap_{x \in m} \phi(x)$ is not a consistent M-modality because $c_{\circlearrowleft} \in \mathcal{S}$ (due to combinatory completeness), and so: $\mathbf{0} \geq \llparenthesis x \leftarrow c_{\circlearrowleft} \cdot c \rrparenthesis \mathbf{0} = \bigsqcap_{x \in \emptyset} \mathbf{0} = \mathbf{1}$, for any code $c \in \mathcal{S}$. Therefore, $\mathbf{1} \leq \mathbf{0}$, which only holds in the trivial complete Heyting prealgebra, where all elements are equivalent.

4.1.3 Realizability Examples

This section illustrates the utility of the MCA framework by providing a few examples of how natural realizability models can be obtained via MCAs. We first show that the standard realizability tripos can be obtained from the PCA-based monadic core (cf. Prop. 6).

▶ **Example 19** (PCAs 🐓)**.** Prop. 6 shows that PCAs correspond to the sub-singleton monad $MA = \{S \subseteq A \mid \forall x_1, x_2 \in S. x_1 = x_2\}$. Following the standard intuitions of realizability models based on PCAs, when applying an evidence of $\phi_1 \xrightarrow{e} \phi_2$ to a code c such that $\phi_1(c)$, the M-modality $\llparenthesis x \leftarrow e \cdot c \rrparenthesis \phi_2$ should express that the computation $e \cdot c$ returns a valid code for ϕ_2, that is, that there exists such a valid code in the corresponding sub-singleton. To that end, we take $\Omega = \mathcal{P}(\star)$ (generalizing the Boolean algebra $\{0,1\}$ making it compatible with intuitionistic meta-theory), and $\llparenthesis x \leftarrow m \rrparenthesis \phi(x) := \bigsqcup_{x \in m} \phi(x)$, for which the whole set $\mathbb{A}$ defines as usual a valid separator. It is then easy to verify that the evidenced frame obtained from Thm. 16 is the expected one described earlier [10].

Next, to further demonstrate the uniformity and utility of our framework, we go back to a couple of our MCA instances from Sec. 3.2. We equip each instance with a corresponding M-modality and demonstrate that this indeed recovers the expected model.

▶ **Example 20** (RCA 🐓)**.** As described in Sec. 3.2.1, relational realizability use the powerset monad $MA = \mathcal{P}(A)$. PCAs are then a special case of RCAs, as the sub-singleton is a special case of the powerset monad. The standard modalities for RCAs are the M-modalities of angelic and demonic nondeterminism, both using $\Omega = \mathcal{P}(\star)$ as in PCAs. For angelic nondeterminism, the modality is the same as the one used for PCAs, $\llparenthesis x \leftarrow m \rrparenthesis \phi(x) := \bigsqcup_{x \in m} \phi(x)$, and $\mathbb{A}$ is always a separator. However, for demonic nondeterminism, we take the infimum rather than the supremum. To allow for a separator, the definition of a modality has to conjoin some "termination" predicate $m \Downarrow$, that implies progress for the elements of the designated separator, yielding the definition: $\llparenthesis x \leftarrow m \rrparenthesis \phi(x) := m \Downarrow \wedge \bigsqcap_{x \in m} \phi(x)$.

▶ **Example 21** (SCA 🐓)**.** As in Sec. 3.2.2, for Σ a preordered set of states, SCAs use the increasing (powerset) state monad, $MA = \{m : \Sigma \to \mathcal{P}(\Sigma \times A) \mid \forall \sigma_0 \in \Sigma, (\sigma_1, x) \in m(\sigma_0). \sigma_0 \leq \sigma_1\}$. To account for state, predicates have an extra component in Σ, so $\Omega = \Sigma \to \mathcal{P}(\star)$, and they are restricted so that they must be "future-stable", i.e. upward closed with respect to states. As in RCAs, there are angelic and demonic modalities: the angelic is $\left(\llparenthesis x \leftarrow m \rrparenthesis \phi(x)\right)^{\sigma} := \bigsqcap_{\sigma' \geq \sigma} \bigsqcup_{(x,\sigma'') \in m(\sigma')} (\phi(x))^{\sigma''}$, and the demonic again uses a "termination" predicate and is given by: $\left(\llparenthesis x \leftarrow m \rrparenthesis \phi(x)\right)^{\sigma} := \bigsqcap_{\sigma' \geq \sigma}. m(\sigma') \Downarrow \wedge \bigsqcap_{(x,\sigma'') \in m(\sigma')} (\phi(x))^{\sigma''}$.

▶ **Example 22** (Continuations 🐓)**.** As observed in Sec. 3.2.3, the continuation monad provides a particular instance of an MCA replaying a (CbV) CPS translation. Since Krivine classical realizability [22] is known to be equivalent to the composition of CPS translation with a standard intuitionistic realizability interpretation [30], we can easily replay this construction in our setting using a CPSCA based on the continuation monad $MA = (A \to R) \to R$. Krivine realizability models crucially realies on a parameter $\perp\!\!\!\perp$, the so-called *pole*, which intuitively contains the computations considered as valid; here, this will simply be a subset

of R. Any pole induces an orthogonality relation between elements of A and of $A \to R$: a function f in the latter is said to be orthogonal to a, written $f \perp a$ when $f(a) \in \perp\!\!\!\perp$.

Since our algebras follow a call-by-value discipline, realizers should be defined using two layers of orthogonality [14, 29]: given a formula A, its interpretation is primitively defined by a set of values $[\![A]\!]$. Then its set of opponents are the continuations k orthogonal to any $a \in [\![A]\!]$, while a realizer will be orthogonal to any such continuation. Last, as is usual in Krivine realizability, as soon as the pole is non-empty, there is at least one continuation k and $c_a \in \mathbb{A}$ such that $k(c_a) \in \perp\!\!\!\perp$. Then, the computation $e_{\mathsf{K}_k} \cdot c_a$, which drops the current continuation and applies instead k to c_a, would be a realizer of $\perp$. To circumvent this issue, we consider the set PL of *proof-like* codes obtained by combinatorial completeness extended with the code e_{cc}. As in [22], a pole $\perp\!\!\!\perp$ is consistent if for any proof-like term m, there exists a continuation k such that $m(k) \notin \perp\!\!\!\perp$. For any such pole, the set PL defines a separator.

Formally, for $\Omega := \mathcal{P}(\star)$, a fixed consistent pole $\perp\!\!\!\perp \subset R$ and $\phi : A \to \Omega$, we define:

$$\langle\!\!\langle\, x \leftarrow m \,\rangle\!\!\rangle\, \phi\,(x) := \bigcap_{k \in A \to R} \left(\left(\bigcap_{a \in A} (\phi(a) \supset k \perp a) \right) \supset m \perp k \right)$$

This definition, which satisfies the expected axiom of an M-modality and admits the set PL as separator, induces via Thm. 16 an evidenced frame corresponding to an indirect-style presentation of a (call-by-value) Krivine realizability model analogous to the one in [14].

▶ **Example 23** (Parameterized 🐓). Bauer [1] provides a construction of a parameterized realizability tripos from a ParCA. The same tripos can be obtained from the MCA representation given in Sec. 3.2.4 following the construction in Cor. 17, taking the following M-modality (given a set of parameters $\mathbf{P}$):

$$\langle\!\!\langle\, x \leftarrow m \,\rangle\!\!\rangle\, \phi\,(x) := \bigcap_{p \in \mathbf{P}} \bigcup_{x \in m(p)} \phi\,(x)\,.$$

The modality extends the one for PCAs by requiring the computation to yield a value in ϕ for every possible parameter in $\mathbf{P}$. Just as in a PCA, here too the set $\mathbb{A}$ provides a separator. Moreover, for the induced logic to be non-trivial the set of parameters $\mathbf{P}$ has to be non-empty.

4.2 Connection to Assembly Models

To further highlight how MCAs naturally generalize PCAs, we here discuss how the construction of assemblies over PCAs, a foundational technique in the study of realizability toposes [44], seamlessly extends to MCAs. Assemblies $\mathbf{Asm}_{\mathcal{A}}$ over a PCA $\mathcal{A}$, are pairs $(X, \|\cdot\|_X)$ where $\|\cdot\|_X$ maps any $x \in X$ to a non-empty subsets of $\mathcal{A}$ witnessing x's existence. The categories of assemblies $\mathbf{Asm}_{\mathcal{A}}$ are somewhat simpler to handle, and sufficient to model rich intuitionistic systems like the Calculus of Construction [4]. In fact, assemblies can be identified as a particular subcategory of the realizability topos, which in itself is a quasi-topos. Moreover, several works studying completions mechanisms within toposes [26, 40] emphasized that the realizability topos could be recovered as the ex/reg completions of $\mathbf{Asm}_{\mathcal{A}}$.

Recently, following a line of work aiming to provide an algebraic counterpart to Krivine realizability using *implicative algebras*, Castro *et al.* extended the usual definition of assemblies to such algebras [5]. While they manage to prove that the resulting assemblies define a quasi-topos as expected, in this context the usual completion mechanism techniques seem to fail and relating assemblies with the corresponding implicative topos remains an open problem to date. Following the connection established between implicative algebras and evidenced frames in [10], we can adapt their construction to define assemblies over any

evidenced frame which, combined with Thm. 16 , provide us with a construction of assemblies over any MCA. Note that assemblies over a PCA or an implicative algebra are then recovered by considering the corresponding evidenced frame.

▶ **Definition 24** (Category $\mathbf{Asm}_{\mathcal{EF}}$). *The category of assemblies over an evidenced frame* $\mathcal{EF} = (\Phi, E, \to)$*, is given by:*

Objects : *an* assembly *over* $\mathcal{EF}$ *is a tuple* $X = (|X|, \mathcal{R}_X)$ *where* $|X|$ *is a set and* $\mathcal{R}_X : |X| \to \Phi_{\mathsf{ev}}$*, where* $\Phi_{\mathsf{ev}} = \{\varphi \in \Phi \mid \exists e \in E. \top \xrightarrow{e} \varphi\}$ *is the subset of evidenced relations.*

Morphisms : *given two assemblies* X, Y*, a* morphism f *from* X *to* Y *is a function* $|f| : |X| \to |Y|$ *s.t. there exists an evidence* $\tau_f \in E$*, s.t for all* $x \in |X|$*,* $\mathcal{R}_X(x) \xrightarrow{\tau_f} \mathcal{R}_Y(f\,(x))$*. That evidence* τ_f *is said to* "track" f*.*

To prove that this indeed defines a category, it suffices to observe that the *identity morphism* over assembly X is simply the identity function $\mathrm{id}_{|X|}$ over $|X|$ tracked by the reflexivity evidence e_{id}; and that if f is tracked by τ_f and g is tracked by τ_g, then their composition $f \circ g$ is the composition of $|f| \circ |g|$ tracked by the transitivity evidence $\tau_g ; \tau_f$. Studying more in-depth the category $\mathbf{Asm}_{\mathcal{EF}}$ is out of the scope of this paper, but we conjecture that one could follow the development for implicative assemblies in [5] to prove that the $\mathbf{Asm}_{\mathcal{EF}}$ is also finitely (co)complete, locally cartesian closed and possesses a strong object classifier, and defines then a quasi-topos. Nonetheless, as for the implicative case, it is not clear whether some mechanism analogous to the ex/reg completion could connect it to the topos induced by the evidenced frame.

5 Conclusion and Future Work

This paper introduces Monadic Combinatory Algebras (MCAs) as a novel extension of PCAs that encompasses a wide range of computational effects through the use of monads. This new framework addresses the limitations of traditional PCAs, which only support non-termination as a computational effect, by providing a more comprehensive model capable of internalizing effects such as nondeterminism, stateful computations, continuations and oracles. We link MCAs to realizability theory (generalizing the role of PCAs in traditional realizability models) by providing two uniform constructions of realizability models from MCAs, triposes and assemblies, that factor through evidenced frames. Overall, MCAs provide a powerful and flexible framework for internalizing computational effects that opens up new avenues in the study of effectful computations and their algebraic and categorical models.

Future research into the MCA framework presents several directions for exploration. A key one is examining how different underlying monads affect the resulting theory, potentially providing novel insights into constructive models of computation. Further work is also needed to extend the MCA scope to unaddressed effects, such as probabilistic computation. Notably, our MCA structure is based on **Set**-monads, whereas probabilistic computation is typically formalized using the Giry monad [15] in the category **Meas** (measurable spaces and functions). Advancing the theory of combinatory objects in Freyd categories will enable the exploration of broader computational effects beyond **Set**-based monads. Moreover, a more comprehensive categorical understanding of MCAs, via their connections to structures like monoidal, enriched, and higher categories, is needed.

The MCA framework can also be further extended to support more diverse and complex notions of computation such as hybrid effects, where multiple monads are combined to model complex interactions between different computational effects. Additionally, the framework can be enriched by incorporating alternative evaluation strategies, such as Call-by-Name, which may uncover new computational behaviors.

References

1 Andrej Bauer and James E. Hanson. The Countable Reals, 2024. URL: https://arxiv.org/abs/2404.01256, arXiv:2404.01256.

2 Stefano Berardi, Marc Bezem, and Thierry Coquand. On the Computational Content of the Axiom of Choice. *The Journal of Symbolic Logic*, 63(2):600–622, 1998. doi:10.2307/2586854.

3 Simon Boulier, Pierre-Marie Pédrot, and Nicolas Tabareau. The Next 700 Syntactical Models of Type Theory. In *Proceedings of the 6th ACM SIGPLAN Conference on Certified Programs and Proofs*, CPP 2017, page 182–194, New York, NY, USA, 2017. Association for Computing Machinery. doi:10.1145/3018610.3018620.

4 Aurelio Carboni, Peter J. Freyd, and Andre Scedrov. A Categorical Approach to Realizability and Polymorphic Types. In M. Main, A. Melton, M. Mislove, and D. Schmidt, editors, *Mathematical Foundations of Programming Language Semantics*, pages 23–42, Berlin, Heidelberg, 1988. Springer Berlin Heidelberg. doi:10.1007/3-540-19020-1_2.

5 Félix Castro, Alexandre Miquel, and Krzysztof Worytkiewicz. Implicative Assemblies, 2023. URL: https://arxiv.org/abs/2304.10429, arXiv:2304.10429.

6 J.R.B. Cockett and P.J.W. Hofstra. Introduction to Turing Categories. *Annals of Pure and Applied Logic*, 156(2):183–209, 2008. doi:10.1016/j.apal.2008.04.005.

7 Robin Cockett and Stephen Lack. Restriction Categories III: Colimits, Partial Limits and Extensivity. *Mathematical Structures in Computer Science*, 17(4):775–817, 2007. doi:10.1017/S0960129507006056.

8 Liron Cohen, Sofia Abreu Faro, and Ross Tate. The Effects of Effects on Constructivism. *Electronic Notes in Theoretical Computer Science*, 347:87–120, 2019. doi:10.1016/j.entcs.2019.09.006.

9 Liron Cohen, Ariel Grunfeld, Dominik Kirst, Étienne Miquey, and Ross Tate. Coq Formalization, https://github.com/dominik-kirst/mca. Supplementary Material, 2025.

10 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, 2021. doi:10.1109/LICS52264.2021.9470514.

11 Olivier Danvy. On Evaluation Contexts, Continuations, and the Rest of the Computation. In *Proceedings of the Fourth ACM SIGPLAN Workshop on Continuations, Technical report CSR-04-1, Department of Computer Science, Queen Mary's College*, pages 13–23, 2004.

12 Solomon Feferman. A Language and Axioms for Explicit Mathematics. In *Algebra and logic*, pages 87–139. Springer, 1975. doi:10.1007/BFb0062852.

13 Jonas Frey. Characterizing Partitioned Assemblies and Realizability Toposes. *Journal of Pure and Applied Algebra*, 223(5):2000–2014, 2019. doi:10.1016/j.jpaa.2018.08.012.

14 Samuel Gardelle and Étienne Miquey. Do CPS Translations Also Translate Realizers? In Timothy Bourke and Delphine Demange, editors, *JFLA 2023 - 34èmes Journées Francophones des Langages Applicatifs*, pages 103–120, Praz-sur-Arly, France, January 2023. URL: https://hal.inria.fr/hal-03910311.

15 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. doi:10.1007/BFb0092872.

16 Kurt Gödel. On Intuitionistic Arithmetic and Number Theory. *Collected Works*, 1:287–295, 1933.

17 Mayer Goldberg. Ellipses and Lambda Definability. *Logical Methods in Computer Science*, Volume 11, Issue 3, October 2015. doi:10.2168/lmcs-11(3:25)2015.

18 Pieter J. W. Hofstra. All Realizability is Relative. *Mathematical Proceedings of the Cambridge Philosophical Society*, 141:239 – 264, 2006. doi:10.1017/S0305004106009352.

19 Pieter JW Hofstra. Partial Combinatory Algebras and Realizability Toposes. *University of Ottowa*, 2004.

20 J. M. E. Hyland, P. T. Johnstone, and A. M. Pitts. Tripos theory. *Mathematical Proceedings of the Cambridge Philosophical Society*, 88(2):205–232, 1980. doi:10.1017/S0305004100057534.

21 Stephen Cole Kleene. On the Interpretation of Intuitionistic Number Theory. *The journal of symbolic logic*, 10(4):109–124, 1945. doi:10.2307/2269016.

22 Jean-Louis Krivine. Realizability in Classical Logic. In Interactive Models of Computation and Program Behaviour. *Panoramas et synthèses*, 27, 2009. URL: https://hal.science/hal-00154500.

23 Jean-Louis Krivine. Realizability Algebras: A Program to Well Order R. *Logical methods in computer science*, 7, 2011. doi:10.2168/LMCS-7(3:2)2011.

24 Paul Blain Levy, John Power, and Hayo Thielecke. Modelling Environments in Call-by-Value Programming Languages. *Information and Computation*, 185(2):182–210, 2003. doi:10.1016/S0890-5401(03)00088-9.

25 Giuseppe Longo and Eugenio Moggi. A Category-theoretic Characterization of Functional Completeness. *Theoretical Computer Science*, 70(2):193–211, 1990. doi:10.1016/0304-3975(90)90122-X.

26 M. Menni. *Exact Completions and Toposes*. PhD thesis, University of Edinburgh, 2000.

27 Eugenio Moggi. Computational Lambda-calculus and Monads. *[1989] Proceedings. Fourth Annual Symposium on Logic in Computer Science*, pages 14–23, 1989. doi:10.1109/LICS.1989.39155.

28 Eugenio Moggi. Notions of Computation and Monads. *Information and computation*, 93(1):55–92, 1991. doi:10.1016/0890-5401(91)90052-4.

29 Guillaume Munch-Maccagnoni. Focalisation and Classical Realisability. In Erich Grädel and Reinhard Kahle, editors, *Computer Science Logic '09*, volume 5771 of *Lecture Notes in Computer Science*, pages 409–423. Springer, Heidelberg, 2009. doi:10.1007/978-3-642-04027-6_30.

30 P. Oliva and T. Streicher. On Krivine's Realizability Interpretation of Classical Second-Order Arithmetic. *Fundam. Inform.*, 84(2):207–220, 2008. doi:10.5555/1402673.1402677.

31 Pierre-Marie Pédrot. Russian Constructivism in a Prefascist Theory. In Holger Hermanns, Lijun Zhang, Naoki Kobayashi, and Dale Miller, editors, *35th Annual ACM/IEEE Symposium on Logic in Computer Science, Saarbrücken, Germany*, pages 782–794. ACM, 2020. doi:10.1145/3373718.3394740.

32 Pierre-Marie Pédrot and Nicolas Tabareau. Failure is Not an Option - An Exceptional Type Theory. In *27th European Symposium on Programming*, volume 10801 of *LNCS*, pages 245–271, Thessaloniki, Greece, April 2018. Springer. doi:10.1007/978-3-319-89884-1_9.

33 Pierre-Marie Pédrot and Nicolas Tabareau. The Fire Triangle: How to Mix Substitution, Dependent Elimination, and Effects. *Proc. ACM Program. Lang.*, 4(POPL):58:1–58:28, 2020. doi:10.1145/3371126.

34 Pierre-Marie Pédrot, Nicolas Tabareau, Hans Jacob Fehrmann, and Éric Tanter. A Reasonably Exceptional Type Theory. *Proc. ACM Program. Lang.*, 3(ICFP):108:1–108:29, 2019. doi:10.1145/3341712.

35 Andrew M Pitts. Evaluation logic. In *IV Higher Order Workshop, Banff 1990: Proceedings of the IV Higher Order Workshop, Banff, Alberta, Canada 10–14 September 1990*, pages 162–189. Springer, 1991. doi:10.1007/978-1-4471-3182-3_11.

36 Andrew M Pitts. Tripos Theory in Retrospect. *Mathematical structures in computer science*, 12(3):265–279, 2002. doi:10.1016/S1571-0661(04)00107-0.

37 John Power. Premonoidal Categories as Categories with Algebraic Structure. *Theoretical Computer Science*, 278(1):303–321, 2002. Mathematical Foundations of Programming Semantics 1996. doi:10.1016/S0304-3975(00)00340-6.

38 John Power and Edmund Robinson. Premonoidal Categories and Notions of Computation. *Mathematical structures in computer science*, 7(5):453–468, 1997. doi:10.1017/S0960129597002375.

39 John Power and Hayo Thielecke. Environments, Continuation Semantics and Indexed Categories. In Martín Abadi and Takayasu Ito, editors, *Theoretical Aspects of Computer Software*, pages 391–414, Berlin, Heidelberg, 1997. Springer Berlin Heidelberg. doi:10.1007/BFb0014560.

40 Edmund Robinson and Giuseppe Rosolini. Colimit Completions and the Effective Topos. *Journal of Symbolic Logic*, 55(2):678–699, 1990. doi:10.2307/2274658.

41 Walter Ferrer Santos, Jonas Frey, Mauricio Guillermo, Octavio Malherbe, and Alexandre Miquel. Ordered Combinatory Algebras and Realizability. *Mathematical Structures in Computer Science*, 27(3):428–458, 2017. doi:10.1017/S0960129515000432.

42 Samuel L. Speight. Groupoidal Realizability for Intensional Type Theory. *Mathematical Structures in Computer Science*, page 1–34, 2024. doi:10.1017/S0960129524000343.

43 Thomas Streicher. Krivine's Classical Realisability from a Categorical Perspective. *Mathematical Structures in Computer Science*, 23(6):1234–1256, 2013. doi:10.1017/S0960129512000989.

44 Jaap Van Oosten. *Realizability: an Introduction to its Categorical Side*, volume 152. Elsevier, Amsterdam, 2008. doi:doi:10.1017/S1079898600000858.

45 Philip Wadler. Comprehending Monads. In *Proceedings of the 1990 ACM Conference on LISP and Functional Programming*, pages 61–78, 1990. doi:10.1145/91556.91592.

46 Philip Wadler. Monads for FunctionalDOI:10.1017/S0960129597002375 Programming. In *International School on Advanced Functional Programming*, pages 24–52. Springer, 1995. doi:10.1007/3-540-59451-5_2.

A Proof of Prop. 6

Proof. A PCA is obtained from an MCA by instantiating the monad with the sub-singleton monad, i.e. MA is the set of subsets of A in which all elements are equal.

$$MA = \{S \subseteq A \mid \forall x_1, x_2 \in S. x_1 = x_2\} \qquad \eta_A(x) = \{x\} \qquad \mu_A(\mathbb{X}) = \bigcup_{X \in \mathbb{X}} X$$

The sub-singleton monad, of sets with at most one element, allows for the interpretation of deterministic binary relations as functions, where the codomain is in the monad. Given a deterministic relation $R \subseteq A \times B$, it is equivalent to a function $\widehat{R} : A \to \{S \subseteq B \mid \forall y_1, y_2 \in S. y_1 = y_2\}$ where $\widehat{R}(a) = \{b\}$ if $R(a, b)$ and $\widehat{R}(a) = \emptyset$ otherwise. Because the evaluation relation is determinstic, using the sub-singleton monad to define an MCA, the MCA laws become the laws of a PCA. That is, instead of using $c_f \cdot c_a \downarrow c$ and $e \downarrow c$ in a PCA, we use $c_f \cdot c_a = \{c\}$ and $\nu(e) = \{c\}$ in the MCA, respectively. We note that, when working in a classical metatheory, PCAs can also be obtained by instantiating MCAs with the maybe monad. ◀

B Proof of Prop. 7

Proof. Let $\mathbb{A}$ be a monadic applicative structure. If $\mathbb{A}$ is an MCA then the e_{S} and e_{K} combinators are the codes $\langle \lambda^2.(0 \cdot 2) \cdot (1 \cdot 2) \rangle$ and $\langle \lambda^1.0 \rangle$ modeling the λ-calculus terms $\lambda x.\lambda y.\lambda z.\,(x\ z)\ (y\ z)$ and $\lambda x.\lambda y.x$, respectively.

Conversely, if $\mathbb{A}$ has e_{S} and e_{K} combinators satisfying the axioms, then for every term $e \in E_{n+1}(\mathbb{A})$, we can define a code $\langle \lambda^n.e \rangle$ using the following bracket abstraction algorithm. First, we define a closed abstraction term $|\lambda^n.e| \in E_0(\mathbb{A})$ for every $n \in \mathbb{N}$ and any $e \in E_{n+1}(\mathbb{A})$ as follows:

$$\begin{array}{l|l} |\lambda^n.0| := K_n & \left|\lambda^{n+1}.j+1\right| := e_{\mathsf{K}} \bullet |\lambda^n.j| \\ |\lambda^n.c| := K_{n+1} \bullet c & |\lambda^n.e_1 \bullet e_2| := S_{n+1} \bullet |\lambda^n.e_1| \bullet |\lambda^n.e_2| \end{array}$$

The definition uses K_n and S_n, which are the n-ary K and S combinators [17], where $B := e_{\mathsf{S}} \bullet (e_{\mathsf{K}} \bullet e_{\mathsf{S}}) \bullet e_{\mathsf{K}}$:

$$\begin{array}{lll} K_0 := e_{\mathsf{S}} \bullet e_{\mathsf{K}} \bullet e_{\mathsf{K}} & K_1 := e_{\mathsf{K}} & K_{n+2} := B \bullet e_{\mathsf{K}} \bullet K_{n+1} \\ S_0 := e_{\mathsf{S}} \bullet e_{\mathsf{K}} \bullet e_{\mathsf{K}} & S_1 := e_{\mathsf{S}} & S_{n+2} := B \bullet e_{\mathsf{S}} \bullet (B \bullet S_{n+1}) \end{array}$$

The axioms ensure there is a code $c_{n,e} \in \mathbb{A}$ for any $n \in \mathbb{N}$ and any $e \in E_{n+1}(\mathbb{A})$ such that $\nu(|\lambda^n.e|) = [c_{n,e}]$, hence we take $\langle \lambda^n.e \rangle$ to be a $c_{n,e}$, for which it is straightforward to verify the MCA conditions. ◀

C Elaborated Sec. 3.3 — Categorical Characterization of MCAs

We first recall the formal definition of Freyd categories and some associated categorical components [37]. We start with binoidal categories, which capture the idea of non-commutativity of general effectful computation. For example, given two computations m_1, m_2 in the state monad (Sec. 3.2.2), then $\mathsf{let}\ x_1 \Leftarrow m_1\ \mathsf{in}\ \mathsf{let}\ x_2 \Leftarrow m_2\ \mathsf{in}\ (x_1, x_2)$ will generally not yield the same computation as $\mathsf{let}\ x_2 \Leftarrow m_2\ \mathsf{in}\ \mathsf{let}\ x_1 \Leftarrow m_1\ \mathsf{in}\ (x_1, x_2)$, because in the former case m_2 depends on the state modified by m_1, while in the latter case m_1 depends on the state modified by m_2, so the order in which they are sequenced matter. Binoidal categories abstract this behavior.

▶ **Definition 1** (Binoidal Category). *A category* $\mathbb{C}$ *is a* binoidal category *if :*

- *For every* $A, B \in \mathbb{C}$*, there is an object* $A \otimes B$
- *For every* $A \in \mathbb{C}$*, there is a functor* $A \rtimes (-)$*, sending morphisms in* $\mathbb{C}(B_1, B_2)$ *to morphisms in* $\mathbb{C}(A \otimes B_1, A \otimes B_2)$
- *For every object* $B \in \mathbb{C}$*, a functor* $(-) \ltimes B$ *, sending morphisms in* $\mathbb{C}(A_1, A_2)$ *to morphisms in* $\mathbb{C}(A_1 \otimes B, A_2 \otimes B)$

A morphism $f : A \to B$ in a binoidal category $\mathbb{K}$ is *central* if for every other morphism $u : X \to Y$ in $\mathbb{K}$, $(u \ltimes B) \circ (X \rtimes f) = (Y \rtimes f) \circ (u \ltimes A)$ and $(f \ltimes Y) \circ (A \rtimes u) = (B \rtimes u) \circ (f \ltimes X)$. Given a binoidal category $\mathbb{K}$, the centre of $\mathbb{K}$ is the subcategory of $\mathbb{K}$ consisting of all the objects of $\mathbb{K}$ and the central morphisms.

▶ **Definition 2** (Symmetric Premonoidal Category). *A* symmetric premonoidal category *is a binoidal category equipped with an object* $\mathbb{I}$ *and the following central natural isomorphisms:*

Associator: $\alpha_{A,B,C} : (A \otimes B) \otimes C \to A \otimes (B \otimes C)$
Left and right unitor: $\lambda_A : \mathbb{I} \otimes A \to A$ *and* $\rho_A : A \otimes \mathbb{I} \to A$
Swap: $\sigma_{A,B} : A \otimes B \to B \otimes A$

The above natural isomorphisms obey the triangle and pentagon coherence laws as their counterparts in a monoidal category.

▶ **Definition 3** (Freyd Category). *A Freyd category is a triple* $\langle \mathbb{C}, \mathbb{K}, J \rangle$ *such that:*

- $\mathbb{C}$ *is a cartesian category*
- $\mathbb{K}$ *is a symmetric premonoidal category with the same objects as* $\mathbb{C}$
- $J : \mathbb{C} \to \mathbb{K}$ *is an identity on objects functor, strictly preserving symmetric premonoidal structure, whose image lies inside the centre of* $\mathbb{K}$*.*

To define *combinatory objects* we extend the $\circledast$ morphism of the applicative object to an n-ary operator. The iterated product A^n of an object A is recursively defined as: $A^0 := \mathbf{1}$, $A^1 := A$, and $A^{n+2} := A \times A^{n+1}$. Then, projections and pairings are similarly generalized. The association α is generalized to normalize pairs of arbitrary iterated products to a single iterated product, with $\alpha^0 := \lambda$, $\alpha^1 := \mathrm{id}$, and α^{n+2} defined as the composition:

$$A^{k+2} \times A^m \xrightarrow{\alpha} A \times \left(A^{k+1} \times A^m\right) \xrightarrow{A \rtimes \alpha^{k+1}} A^{m+k+2}$$

Finally, $\circledast^n \in \mathbb{K}\left(\mathbb{A}^{n+1}, \mathbb{A}\right)$ is defined by: $\circledast^0 := \rho$, $\circledast^1 := \circledast$, and taking $\circledast^{n+2}$ to be:

$$\mathbb{A} \times \left(\mathbb{A} \times \mathbb{A}^{n+1}\right) \xrightarrow{\alpha^{-1}} (\mathbb{A} \times \mathbb{A}) \times \mathbb{A}^{n+1} \xrightarrow{\circledast \ltimes \mathbb{A}^{n+1}} \mathbb{A} \times \mathbb{A}^{n+1} \xrightarrow{\circledast^{n+1}} \mathbb{A}$$

▶ **Definition 4** ($\mathbb{A}$-monomials). *Given a applicative object $\mathbb{A}$, $\mathbb{A}$-monomials are defined using expressions similar to the ones in Sec. 2.1:*

$$p ::= i \in \mathbb{N} \mid c \in \mathbb{C}\left(\mathbf{1}, \mathbb{A}\right) \mid p \bullet p \qquad\qquad E_n\left(\mathbb{A}\right) ::= \{p \mid \textit{all is in } p \textit{ are } < n\}$$

The evaluation function $[\![-]\!]_n : E_n\left(\mathbb{A}\right) \to \mathbb{K}\left(\mathbb{A}^n, \mathbb{A}\right)$ relates expressions to $\mathbb{K}$-morphisms:

$$\begin{aligned}
[\![i]\!]_n &:= J\pi^n_{i+1} \\
[\![c]\!]_n &:= J\left(c \circ !\right) \\
[\![p_f \bullet p_a]\!]_n &:= \circledast \circ \left(\mathbb{A} \rtimes [\![p_a]\!]_n\right) \circ \left([\![p_f]\!]_n \ltimes \mathbb{A}^n\right) \circ J\Delta
\end{aligned}$$

An $\mathbb{A}$-monomial is a morphism f in $\mathbb{K}$ s.t there exists $n \in \mathbb{N}$ and a term $p_f \in E_n\left(\mathbb{A}\right)$ for which $[\![p_f]\!]_n = f$.

▶ **Lemma C5.** *For all $n \in \mathbb{N}$, all $p \in E_n\left(\mathbb{A}\right)$, and all $c_1, \ldots, c_n \in \mathbb{C}\left(\mathbf{1}, \mathbb{A}\right)$:*

$$[\![p\{c_1\}\cdots\{c_n\}]\!]_0 = [\![p]\!]_n \circ J\left\langle c_1, \ldots, c_n\right\rangle_n$$

Proof. By structural induction on p. ◀

▶ **Proposition 6.** *Let $\mathbb{A}$ be an applicative object. $\mathbb{A}$ is a combinatory object iff the k and s morphisms in $\mathbb{K}$ defined below are $\mathbb{A}$-computable.*

$$\begin{aligned}
&\mathsf{k} : \mathbb{A}^2 \to \mathbb{A} \quad \mathsf{k} := J\pi_1 \\
&\mathsf{s} : \mathbb{A}^3 \to \mathbb{A} \quad \mathsf{s} := \circledast \circ \left(\mathbb{A} \rtimes \circledast\right) \circ \left(\circledast \ltimes \mathbb{A}^2\right) \circ J\left\langle \left\langle \pi^3_1, \pi^3_3 \right\rangle, \left\langle \pi^3_2, \pi^3_3 \right\rangle \right\rangle
\end{aligned}$$

Proof. Let $\mathbb{A}$ be an applicative object. Assuming all positive $\mathbb{A}$-monomials are $\mathbb{A}$-computable, we consider the expressions $p_{\mathsf{k}} \in E_2\left(\mathbb{A}\right)$ and $p_{\mathsf{s}} \in E_3\left(\mathbb{A}\right)$:

$$p_{\mathsf{k}} := 0 \qquad p_{\mathsf{s}} := \left(0 \bullet 2\right) \bullet \left(1 \bullet 2\right)$$

Since $[\![p_{\mathsf{k}}]\!]_2 = \mathsf{k}$ and $[\![p_{\mathsf{s}}]\!]_3 = \mathsf{s}$, then k and s are $\mathbb{A}$-monomials, and thus $\mathbb{A}$-computable.

Conversely, if k and s are $\mathbb{A}$-computable, let e_{K} and e_{S} be their corresponding codes (with their associated codes of partial applications). For every term $p \in E_{n+1}\left(\mathbb{A}\right)$, we define a code $\langle\lambda^n.p\rangle \in \mathbb{C}\left(\mathbf{1}, \mathbb{A}\right)$ using the following bracket abstraction algorithm. First, for every $p \in E_{n+1}\left(\mathbb{A}\right)$, we define a closed abstraction expression $|\lambda^n.p| \in E_0\left(\mathbb{A}\right)$ as follows:

$$\begin{array}{l|l}
|\lambda^n.0| := K_n & \left|\lambda^{n+1}.j+1\right| := e_{\mathsf{K}} \bullet |\lambda^n.j| \\
|\lambda^n.c| := K_{n+1} \bullet c & |\lambda^n.p_1 \bullet p_2| := S_{n+1} \bullet |\lambda^n.p_1| \bullet |\lambda^n.p_2|
\end{array}$$

The definition uses K_n and S_n, which are the n-ary K and S combinators [17], where $B := e_{\mathsf{S}} \bullet \left(e_{\mathsf{K}} \bullet e_{\mathsf{S}}\right) \bullet e_{\mathsf{K}}$:

$$\begin{array}{lll}
K_0 := e_{\mathsf{S}} \bullet e_{\mathsf{K}} \bullet e_{\mathsf{K}} & K_1 := e_{\mathsf{K}} & K_{n+2} := B \bullet e_{\mathsf{K}} \bullet K_{n+1} \\
S_0 := e_{\mathsf{S}} \bullet e_{\mathsf{K}} \bullet e_{\mathsf{K}} & S_1 := e_{\mathsf{S}} & S_{n+2} := B \bullet e_{\mathsf{S}} \bullet \left(B \bullet S_{n+1}\right)
\end{array}$$

Since k and s are $\mathbb{A}$-computable, the axioms ensure that for every $p \in E_{n+1}\left(\mathbb{A}\right)$ there is a code $\langle\lambda^n.p\rangle \in \mathbb{C}\left(\mathbf{1}, \mathbb{A}\right)$ such that $[\![|\lambda^n.p|]\!]_0 = J\left\langle\lambda^n.p\right\rangle$, and for all $c_1, \ldots, c_k \in \mathbb{C}\left(\mathbf{1}, \mathbb{A}\right)$ (where $k < n$) the following hold:

$$[\![\left\langle\lambda^{n-k}.p\{c_1\}\cdots\{c_k\}\right\rangle \bullet 0 \bullet \cdots \bullet n-k-1]\!]_{n-k} = [\![p\{c_1\}\cdots\{c_k\}]\!]_{n-k} \tag{1}$$

$$[\![\langle\lambda^n.p\rangle \bullet c_1 \bullet \cdots \bullet c_k]\!]_0 = [\![\left\langle\lambda^{n-k}.p\{c_1\}\cdots\{c_k\}\right\rangle]\!]_0 \tag{2}$$

Using Lem. C5, the above equations correspond to the square and triangle diagrams of Def. 9. Hence we take $e_{f(c_1,\ldots,c_k)}$ to be a $\left\langle\lambda^{n-k}.p_f\{c_1\}\cdots\{c_k\}\right\rangle$, for which it is straightforward to verify the combinatory object conditions. ◀

Next we relate the set-based MCAs to the abstract categorical notion of combinatory objects. For this, we recall that the Kleisli category of every strong monad is a premonoidal category [38]. Here, we work in **Set**, where every monad is strong. Together, this entails that for every monad, the Kleisli category gives rise to a Freyd category. Since the details of the construction are relevant to our result, we here provide them below.

▶ **Lemma C7.** *Given a* **Set** *monad* M*, the triple* $\langle \mathbf{Set}, \mathbf{Set}_M, J_M \rangle$ *forms a Freyd category, where* $\mathbf{Set}_M$ *is the Kleisli category of* M *over* **Set***, and* $J_M : \mathbf{Set} \to \mathbf{Set}_M$ *is the canonical functor defined by* $J_M(f) = \eta \circ f$*.*

Proof. Given a monad M, the constructs making $(\mathbf{Set}, \mathbf{Set}_M, J_M)$ a Freyd category are:

- The cartesian structure of **Set** is given by the cartesian product of sets $\times$
- For every function $f : A_1 \to MA_2$: $(f \ltimes B)(x_1, y) = \mathsf{let}\ x_2 \Leftarrow f(x_1)\ \mathsf{in}\ [(x_2, y)]$
- For every function $f : B_1 \to MB_2$: $(A \rtimes f)(x, y_1) = \mathsf{let}\ y_2 \Leftarrow f(y_1)\ \mathsf{in}\ [(x, y_2)]$
- The associator, left unitor, right unitor, and swap are given by composing η on the associator, left unitor, right unitor, and swap of the underlying cartesian structure of **Set**.

The Freyd requirements for this construction are easily verified. ◀

Since the application morphism of an applicative system is a morphism in $\mathbb{K}$, which is the category of computations in a Freyd category, it corresponds to the Kleisli application of an MAS.

▶ **Lemma C8.** *Let* M *be a* **Set** *monad.* $\mathbb{A}$ *is a MAS over* M *if and only if it is an applicative object in* $\mathbf{Set}_M$*.*

Proof. The application morphism of the applicative object is the application Kleisli function of the MAS: $\circledast(c_1, c_2) = c_1 \cdot c_2$. ◀

▶ **Theorem 11.** *Let* M *be a* **Set** *monad.* $\mathbb{A}$ *is an MCA over* M *if and only if it is a combinatory object in* $\mathbf{Set}_M$*.*

Proof. We use the s and k formulation of an MCA given in Prop. 7 to prove the equivalence to s and k being $\mathbb{A}$-computable. Using the same set of codes $e_{\mathsf{S}}, e_{\mathsf{K}}, e_{\mathsf{S}(c_1)}, e_{\mathsf{S}(c_1,c_2)}$, and $e_{\mathsf{K}(c_1)}$ for any two codes c_1, c_2, it is easy to verify that the equations in Prop. 7 are equivalent to those in Def. 9. ◀

The categorical definition of a PCA-object in Turing categories relies on a specific notion of products, which, in turn, relies on a general notion of restrictions [6]. A restriction structure is a convenient way of handling partiality in category theory. Roughly speaking, considering morphisms as partial maps, a restriction assigns to each partial map f a partial identity map $\overline{f}$ with the same domain as f. To relate to that definition, we here show that cartesian restriction categories are a special case of Freyd categories.

▶ **Definition 9** (Restriction structure)**.** *A* restriction structure *on a category* $\mathbb{C}$ *assigns to every morphism* $f : A \to B$ *a morphism* $\overline{f} : A \to A$ *s.t. the following hold:*

- *For* $f : A \to B$*:* $f \circ \overline{f} = f$
- *For* $f : A \to B_1$ *and* $g : A \to B_2$*:* $\overline{f} \circ \overline{g} = \overline{g} \circ \overline{f}$ *and* $\overline{f \circ \overline{g}} = \overline{f} \circ \overline{g}$
- *For* $f : B \to C$ *and* $g : A \to B$*:* $\overline{f} \circ g = g \circ \overline{f \circ g}$

A morphism f is *total* when $\overline{f} = \mathrm{id}$. The total morphsims in a category $\mathbb{C}$ form a subcategory of $\mathbb{C}$, called $\mathrm{Tot}\,(\mathbb{C})$. A restriction in $\mathbb{C}$ induces a partial order on the hom-sets of $\mathbb{C}$ as follows $f \leq g \iff f = g \circ \overline{f}$. A *cartesian restriction category* is a category with a restriction structure along with restriction cartesian products and restriction terminal objects which are weaker variants of the standard notions that still form a monoidal category [7]. In a cartesian restriction category $\mathbb{C}$, the subcategory $\mathrm{Tot}\,(\mathbb{C})$ is a cartesian subcategory of $\mathbb{C}$.

▶ **Proposition 10.** *Cartesian restriction categories induce Freyd categories.*

Proof. Let $\mathbb{C}$ be a cartesian restriction categories. The induced Freyd category is defined by $\langle \mathrm{Tot}\,(\mathbb{C}), \mathbb{C}, J \rangle$ where J is the inclusion functor of $\mathrm{Tot}\,(\mathbb{C})$ into $\mathbb{C}$. It is straightforward to verify that it indeed forms a Freyd category. ◀

The definition of a PCA-object in a cartesian restriction category [6], using S and K combinators, is the exact counterpart of our definition of a combinatory object in a Freyd category, by merely replacing morphisms of a Freyd category with their counterparts in a cartesian restriction category. In fact, the additional requirement needed in that formulation, namely that the code morphism has to be total, is subsumed by taking it from the underlying cartesian category through the functor given in the definition of the Freyd category.

▶ **Proposition 11.** *In Freyd categories induced by cartesian restriction categories, combinatory objects are PCA-objects, as defined in [6].*

D Elaborated Sec. 4.1.2: From Effect Shells to Evidenced Frames

We start by providing a series of useful lemmas.

▶ **Lemma D12.** *For all A, $\phi_1, \phi_2 : A \to \Omega$, and $m \in M(A)$:*

$$\frac{\phi_1(x) \leq \phi_2(x)}{\langle\!\langle x \leftarrow m \rangle\!\rangle \phi_1(x) \leq \langle\!\langle x \leftarrow m \rangle\!\rangle \phi_2(x)}$$

Proof. Assume for all $x \in A$, $\phi_1(x) \leq \phi_2(x)$. By currying, we get:

$$\mathbf{1} \leq \phi_1(x) \sqsupset \phi_2(x)$$

Since this holds for all $x \in A$, then:

$$\mathbf{1} \leq \bigsqcap_{x \in A} (\phi_1(x) \sqsupset \phi_2(x)) \underset{(13)}{\leq} \langle\!\langle x \leftarrow m \rangle\!\rangle \phi_1(x) \sqsupset \langle\!\langle x \leftarrow m \rangle\!\rangle \phi_2(x)$$

From which, by uncurrying, we obtain:

$$\langle\!\langle x \leftarrow m \rangle\!\rangle \phi_1(x) \leq \langle\!\langle x \leftarrow m \rangle\!\rangle \phi_2(x).$$

◀

▶ **Lemma D13.** *For all A, $\phi : A \to \Omega$, $\theta \in \Omega$, and $m \in M(A)$:*

1. $\langle\!\langle x \leftarrow m \rangle\!\rangle (\theta \sqsupset \phi(x)) \leq \theta \sqsupset \langle\!\langle x \leftarrow m \rangle\!\rangle \phi(x)$
2. $\theta \sqcap \langle\!\langle x \leftarrow m \rangle\!\rangle \phi(x) \leq \langle\!\langle x \leftarrow m \rangle\!\rangle (\theta \sqcap \phi(x))$

Proof.

1. By reflexivity, we have:

   $$\forall x \in A. \theta \sqsupset \phi(x) \leq \theta \sqsupset \phi(x)$$

   by uncurrying θ and then currying $\theta \sqsupset \phi(x)$, we get:

   $$\forall x \in A. \theta \leq (\theta \sqsupset \phi(x)) \sqsupset \phi(x)$$

   so by the infimum property we get:

   $$\theta \leq \bigsqcap_{x \in A} ((\theta \sqsupset \phi(x)) \sqsupset \phi(x)) \underset{(13)}{\leq} \langle\!\langle x \leftarrow m \rangle\!\rangle (\theta \sqsupset \phi(x)) \sqsupset \langle\!\langle x \leftarrow m \rangle\!\rangle \phi(x)$$

   Now, by uncurrying $\langle\!\langle x \leftarrow m \rangle\!\rangle (\theta \sqsupset \phi(x))$ and currying θ we get:

   $$\langle\!\langle x \leftarrow m \rangle\!\rangle (\theta \sqsupset \phi(x)) \leq \theta \sqsupset \langle\!\langle x \leftarrow m \rangle\!\rangle \phi(x)$$

2. By reflexivity, we have:

   $$\forall x \in A. \theta \sqcap \phi(x) \leq \theta \sqcap \phi(x)$$

   by currying $\phi(x)$ we get:

   $$\forall x \in A. \theta \leq \phi(x) \sqsupset (\theta \sqcap \phi(x))$$

   so by the infimum property we get:

   $$\theta \leq \bigsqcap_{x \in A} (\phi(x) \sqsupset (\theta \sqcap \phi(x))) \underset{(13)}{\leq} \langle\!\langle x \leftarrow m \rangle\!\rangle \phi(x) \sqsupset \langle\!\langle x \leftarrow m \rangle\!\rangle (\theta \sqcap \phi(x))$$

   Now, by uncurrying $\langle\!\langle x \leftarrow m \rangle\!\rangle \phi(x)$ we get:

   $$\theta \sqcap \langle\!\langle x \leftarrow m \rangle\!\rangle \phi(x) \leq \langle\!\langle x \leftarrow m \rangle\!\rangle (\theta \sqcap \phi(x))$$

◀

▶ **Theorem 16** (Evidenced Frame over Monadic Core 🦫). *Let* $\mathcal{MC}_M = (\mathbb{A}, \Omega, \Diamond, \mathcal{S})$ *be a monadic core. The triple* $\left(\Omega^{\mathbb{A}}, \mathcal{S}, \cdot \dot{\to} \cdot\right)$ *forms an evidenced frame, where*

$$\phi_1 \overset{e}{\to} \phi_2 := \forall c \in \mathbb{A}.\phi_1\,(c) \leq \langle\!\langle\, r \leftarrow e \cdot c \,\rangle\!\rangle\, \phi_2\,(r)\,.$$

Proof. We define the logical and program components and show that they satisfy the required properties in each case. We use the abbreviations: $e_{\mathsf{p}_1} := \langle \lambda^1.0 \rangle$ and $e_{\mathsf{p}_2} := \langle \lambda^1.1 \rangle$.

Reflexivity: Take $e_{\mathtt{id}} := \langle \lambda^0.0 \rangle$. We need to show that

$$\forall c \in \mathbb{A}.\phi\,(c) \leq \langle\!\langle\, r \leftarrow e_{\mathsf{id}} \cdot c \,\rangle\!\rangle\, \phi\,(r)\,.$$

For $c \in \mathbb{A}$:

$$\phi\,(c) \leq \langle\!\langle\, r \leftarrow [c] \,\rangle\!\rangle\, \phi\,(r) = \langle\!\langle\, r \leftarrow \langle \lambda^0.0 \rangle \cdot c \,\rangle\!\rangle\, \phi\,(r) = \langle\!\langle\, r \leftarrow e_{\mathsf{id}} \cdot c \,\rangle\!\rangle\, \phi\,(r)$$

Transitivity: Take $e_1\,;e_2 := \langle \lambda^0.e_2 \bullet (e_1 \bullet 0) \rangle$. We need to show:

$$\frac{\forall c \in \mathbb{A}.\phi_1\,(c) \leq \langle\!\langle\, r \leftarrow e_s \cdot c \,\rangle\!\rangle\, \phi_2\,(r) \quad \forall c \in \mathbb{A}.\phi_2\,(c) \leq \langle\!\langle\, r \leftarrow e_t \cdot c \,\rangle\!\rangle\, \phi_3\,(r)}{\forall c \in \mathbb{A}.\phi_1\,(c) \leq \langle\!\langle\, r \leftarrow e_{e_s;e_t} \cdot c \,\rangle\!\rangle\, \phi_3\,(r)}$$

Given the assumptions, for $c \in \mathbb{A}$:

$$\begin{array}{cl} & \phi_1\,(c) \\ \leq & \langle\!\langle\, s \leftarrow e_s \cdot c \,\rangle\!\rangle\, \phi_2\,(s) \\ \leq & \langle\!\langle\, s \leftarrow e_s \cdot c \,\rangle\!\rangle\, \langle\!\langle\, r \leftarrow e_t \cdot s \,\rangle\!\rangle\, \phi_3\,(r) \\ \underset{Lem.\ D12}{\leq} & \langle\!\langle\, r \leftarrow (\mathsf{let}\ s\ \Leftarrow e_s \cdot c\ \mathsf{in}\ e_t \cdot s) \,\rangle\!\rangle\, \phi_3\,(r) \\ \underset{\text{A.Bind}}{=} & \langle\!\langle\, r \leftarrow \langle \lambda^0.e_t \bullet (e_s \bullet 0) \rangle \cdot c \,\rangle\!\rangle\, \phi_3\,(r) = \langle\!\langle\, r \leftarrow e_{e_s;e_t} \cdot c \,\rangle\!\rangle\, \phi_3\,(r) \end{array}$$

Top: Take $\top := \lambda e.\mathbf{1}$ and $e_\top := e_{\mathtt{id}}$. We need to show:

$$\forall c \in \mathbb{A}.\phi\,(c) \leq \langle\!\langle\, r \leftarrow e_{\mathsf{id}} \cdot c \,\rangle\!\rangle\, \top\,(r)$$

For $c \in \mathbb{A}$:

$$\phi\,(c) \leq \mathbf{1} \underset{\text{A.Return}}{\leq} \langle\!\langle\, r \leftarrow [c] \,\rangle\!\rangle\, \mathbf{1} = \langle\!\langle\, r \leftarrow [c] \,\rangle\!\rangle\, \top\,(r) = \langle\!\langle\, r \leftarrow e_{\mathsf{id}} \cdot c \,\rangle\!\rangle\, \top\,(r)$$

Conjunction: Take:

$$\begin{array}{rl} (\phi_1 \wedge \phi_2)\,(e) := & \langle\!\langle\, c_1 \leftarrow e \cdot e_{\mathsf{p}_1} \,\rangle\!\rangle\, \phi_1\,(c_1) \sqcap \langle\!\langle\, c_2 \leftarrow e \cdot e_{\mathsf{p}_2} \,\rangle\!\rangle\, \phi_2\,(c_2) \\ \langle\!\langle e_1, e_2 \rangle\!\rangle := & \langle \lambda^1.1 \bullet (e_1 \bullet 0) \bullet (e_2 \bullet 0) \rangle \\ e_{\mathtt{fst}} := & \langle \lambda^0.0 \bullet e_{\mathsf{p}_1} \rangle \\ e_{\mathtt{snd}} := & \langle \lambda^0.0 \bullet e_{\mathsf{p}_2} \rangle \end{array}$$

For *intro*, we need to show:

$$\frac{\forall c \in \mathbb{A}.\phi\,(c) \leq \langle\!\langle\, r \leftarrow e_1 \cdot c \,\rangle\!\rangle\, \phi_1\,(r) \quad \forall c \in \mathbb{A}.\phi\,(c) \leq \langle\!\langle\, r \leftarrow e_2 \cdot c \,\rangle\!\rangle\, \phi_2\,(r)}{\forall c \in \mathbb{A}.\phi\,(c) \leq \langle\!\langle\, r \leftarrow \langle\!\langle e_1, e_2 \rangle\!\rangle \cdot c \,\rangle\!\rangle\, (\phi_1 \wedge \phi_2)\,(r)}$$

By combining the assumptions, we get, for any $c \in \mathbb{A}$:

$$\begin{array}{cl} & \phi\,(c) \\ \leq & \langle c_2 \leftarrow e_2 \cdot c \rangle\, \phi_2\,(c_2) \sqcap \langle c_1 \leftarrow e_1 \cdot c \rangle\, \phi_1\,(c_1) \\ \underset{Lem.\ D13}{\leq} & \langle c_1 \leftarrow e_1 \cdot c \rangle\, \left(\langle c_2 \leftarrow e_2 \cdot c \rangle\, \phi_2\,(c_2) \sqcap \phi_1\,(c_1)\right) \\ \underset{Lem.\ D12}{\leq} & \langle c_1 \leftarrow e_1 \cdot c \rangle\, \left(\phi_1\,(c_1) \sqcap \langle c_2 \leftarrow e_2 \cdot c \rangle\, \phi_2\,(c_2)\right) \\ \underset{Lem.\ D12+Lem.\ D13}{\leq} & \langle c_1 \leftarrow e_1 \cdot c \rangle\, \left(\langle c_2 \leftarrow e_2 \cdot c \rangle\, \left(\phi_1\,(c_1) \sqcap \phi_2\,(c_2)\right)\right) \end{array}$$

Now, by using Lem. D12 twice over each of the projections of $\sqcap$, we get:

1. $\phi\,(c) \leq \langle c_1 \leftarrow e_1 \cdot c \rangle\, \langle c_2 \leftarrow e_2 \cdot c \rangle\, \phi_1\,(c_1)$
2. $\phi\,(c) \leq \langle c_1 \leftarrow e_1 \cdot c \rangle\, \langle c_2 \leftarrow e_2 \cdot c \rangle\, \phi_2\,(c_2)$

Using (1) we obtain:

$$\begin{array}{cl} & \phi\,(c) \\ \leq & \langle c_1 \leftarrow e_1 \cdot c \rangle\, \langle c_2 \leftarrow e_2 \cdot c \rangle\, \phi_1\,(c_1) \\ \underset{Lem.\ D12+\text{A.Return}}{\leq} & \langle c_1 \leftarrow e_1 \cdot c \rangle\, \langle c_2 \leftarrow e_2 \cdot c \rangle\, \langle r_1 \leftarrow [c_1] \rangle\, \phi_1\,(r_1) \\ \underset{Lem.\ D12+\text{A.Bind}}{\leq} & \langle c_1 \leftarrow e_1 \cdot c \rangle\, \langle r_1 \leftarrow (\mathsf{let}\ c_2 \ \Leftarrow e_2 \cdot c\ \mathsf{in}\ [c_1]) \rangle\, \phi_1\,(r_1) \\ \underset{\text{A.Bind}}{\leq} & \langle r_1 \leftarrow (\mathsf{let}\ c_1 \ \Leftarrow e_1 \cdot c\ \mathsf{in}\ \mathsf{let}\ c_2 \ \Leftarrow e_2 \cdot c\ \mathsf{in}\ [c_1]) \rangle\, \phi_1\,(r_1) \\ = & \langle r_1 \leftarrow \langle \lambda^0.0 \bullet (e_1 \bullet c) \bullet (e_2 \bullet c) \rangle \cdot \langle \lambda^1.0 \rangle \rangle\, \phi_1\,(r_1) \\ = & \langle r_1 \leftarrow \langle \lambda^0.0 \bullet (e_1 \bullet c) \bullet (e_2 \bullet c) \rangle \cdot e_{\mathsf{p}_1} \rangle\, \phi_1\,(r_1) \end{array}$$

Similarly, from (2) we obtain:

$$\phi\,(c) \leq \langle r_2 \leftarrow \langle \lambda^0.0 \bullet (e_1 \bullet c) \bullet (e_2 \bullet c) \rangle \cdot e_{\mathsf{p}_2} \rangle\, \phi_2\,(r_2)$$

Combining both, we get:

$$\begin{array}{cl} & \phi\,(c) \\ \leq & \quad \langle r_1 \leftarrow \langle \lambda^0.0 \bullet (e_1 \bullet c) \bullet (e_2 \bullet c) \rangle \cdot e_{\mathsf{p}_1} \rangle\, \phi_1\,(r_1) \\ & \sqcap \langle r_2 \leftarrow \langle \lambda^0.0 \bullet (e_1 \bullet c) \bullet (e_2 \bullet c) \rangle \cdot e_{\mathsf{p}_2} \rangle\, \phi_2\,(r_2) \\ = & (\phi_1 \wedge \phi_2)\left(\langle \lambda^0.0 \bullet (e_1 \bullet c) \bullet (e_2 \bullet c) \rangle\right) \\ \underset{\text{A.Return}}{\leq} & \langle r \leftarrow \left[\langle \lambda^0.0 \bullet (e_1 \bullet c) \bullet (e_2 \bullet c) \rangle\right] \rangle\, (\phi_1 \wedge \phi_2)\,(r) \\ = & \langle r \leftarrow \langle\!| e_1, e_2 |\!\rangle \cdot c \rangle\, (\phi_1 \wedge \phi_2)\,(r) \end{array}$$

For *elim1* we need to show:

$$\forall c \in \mathbb{A}.\, (\phi_1 \wedge \phi_2)\,(c) \leq \langle r \leftarrow e_{\mathsf{fst}} \cdot c \rangle\, \phi_1\,(r)$$

For $c \in \mathbb{A}$:

$$\begin{array}{cl} & (\phi_1 \wedge \phi_2)\,(c) \\ = & \langle c_1 \leftarrow c \cdot e_{\mathsf{p}_1} \rangle\, \phi_1\,(c_1) \sqcap \langle c_2 \leftarrow c \cdot e_{\mathsf{p}_2} \rangle\, \phi_2\,(c_2) \leq \langle c_1 \leftarrow c \cdot e_{\mathsf{p}_1} \rangle\, \phi_1\,(c_1) \\ = & \langle c_1 \leftarrow \langle \lambda^0.0 \bullet e_{\mathsf{p}_1} \rangle \cdot c \rangle\, \phi_1\,(c_1) \\ = & \langle c_1 \leftarrow e_{\mathsf{fst}} \cdot c \rangle\, \phi_1\,(c_1) \end{array}$$

The proof of *elim2* is analogous.

Universal Implication Take:

$$\begin{array}{c} \phi \supset \vec{\phi}\,(e) := \bigsqcap_{\phi \in \vec{\phi}} \bigsqcap_{c \in \mathbb{A}} \left(\phi\,(c) \sqsupset \langle r \leftarrow e \cdot c \rangle\, \phi\,(r)\right) \\ \lambda e := \left\langle \lambda^1.e \bullet \left(\left\langle \lambda^2.2 \bullet 0 \bullet 1 \right\rangle \bullet 0 \bullet 1\right) \right\rangle \\ e_{\mathsf{eval}} := \left\langle \lambda^0.e_{\mathsf{id}} \bullet (0 \bullet e_{\mathsf{p}_1}) \bullet (0 \bullet e_{\mathsf{p}_2}) \right\rangle \end{array}$$

For *intro*, we need to show:

$$\frac{\forall \phi_3 \in \vec{\phi} \forall c \in \mathbb{A}.\, (\phi_1 \wedge \phi_2)\,(c) \leq \langle\!\!\langle r \leftarrow e \cdot c \rangle\!\!\rangle\, \phi_3\,(r)}{\forall c \in \mathbb{A}. \phi_1\,(c) \leq \langle\!\!\langle r \leftarrow \lambda e \cdot c \rangle\!\!\rangle \left(\phi_2 \supset \vec{\phi}\right)(r)}$$

Expanding the assumption, we get for all $\phi_3 \in \vec{\phi}$, and $c \in \mathbb{A}$:

$$\langle\!\!\langle c_1 \leftarrow c \cdot e_{\mathsf{p}_1} \rangle\!\!\rangle\, \phi_1\,(c_1) \sqcap \langle\!\!\langle c_2 \leftarrow c \cdot e_{\mathsf{p}_2} \rangle\!\!\rangle\, \phi_2\,(c_2) \leq \langle\!\!\langle r \leftarrow e \cdot c \rangle\!\!\rangle\, \phi_3\,(r)$$

Instantiating with $c = \langle c_1, c_2 \rangle$, for which we have $\langle c_1, c_2 \rangle \cdot e_{\mathsf{p}_1} = [c_1]$ and $\langle c_1, c_2 \rangle \cdot e_{\mathsf{p}_2} = [c_2]$, obtains:

$$\langle\!\!\langle c_1' \leftarrow [c_1] \rangle\!\!\rangle\, \phi_1\,(c_1') \sqcap \langle\!\!\langle c_2' \leftarrow [c_2] \rangle\!\!\rangle\, \phi_2\,(c_2') \leq \langle\!\!\langle r \leftarrow e \cdot \langle c_1, c_2 \rangle \rangle\!\!\rangle\, \phi_3\,(r)$$

From this, using after-ret, we get:

$$\phi_1\,(c_1) \sqcap \phi_2\,(c_2) \leq \langle\!\!\langle r \leftarrow e \cdot \langle c_1, c_2 \rangle \rangle\!\!\rangle\, \phi_3\,(r)$$

By currying $\phi_2\,(c_2)$ we get:

$$\phi_1\,(c_1) \leq \phi_2\,(c_2) \sqsupset \langle\!\!\langle r \leftarrow e \cdot \langle c_1, c_2 \rangle \rangle\!\!\rangle\, \phi_3\,(r)$$

Since this holds for all $\phi_3 \in \vec{\phi}$ and $c_2 \in \mathbb{A}$, we get, by the infimum property, for all $c_1 \in \mathbb{A}$:

$$\begin{array}{ll} & \phi_1\,(c_1) \\ \leq & \bigsqcap_{\phi_3 \in \vec{\phi},\, c_2 \in \mathbb{A}} \left(\phi_2\,(c_2) \sqsupset \langle\!\!\langle r \leftarrow e \cdot \langle c_1, c_2 \rangle \rangle\!\!\rangle\, \phi_3\,(r)\right) \\ = & \bigsqcap_{\phi_3 \in \vec{\phi},\, c_2 \in \mathbb{A}} \left(\phi_2\,(c_2) \sqsupset \langle\!\!\langle r \leftarrow \nu \left(e \bullet \left(\left\langle \lambda^2.2 \bullet 0 \bullet 1 \right\rangle \bullet c_1 \bullet c_2\right)\right) \rangle\!\!\rangle\, \phi_3\,(r)\right) \\ = & \bigsqcap_{\phi_3 \in \vec{\phi},\, c_2 \in \mathbb{A}} \left(\phi_2\,(c_2) \sqsupset \langle\!\!\langle r \leftarrow \left\langle \lambda^0.e \bullet \left(\left\langle \lambda^2.2 \bullet 0 \bullet 1 \right\rangle \bullet c_1 \bullet 0\right)\right\rangle \cdot c_2 \rangle\!\!\rangle\, \phi_3\,(r)\right) \\ \underset{\text{A.Return}}{\leq} & \langle\!\!\langle f \leftarrow \left[\left\langle \lambda^0.e \bullet \left(\left\langle \lambda^2.2 \bullet 0 \bullet 1 \right\rangle \bullet c_1 \bullet 0\right)\right\rangle\right] \rangle\!\!\rangle \bigsqcap_{\phi_3 \in \vec{\phi},\, c_2 \in \mathbb{A}} \left(\phi_2\,(c_2) \sqsupset \langle\!\!\langle r \leftarrow f \cdot c_2 \rangle\!\!\rangle\, \phi_3\,(r)\right) \\ = & \langle\!\!\langle f \leftarrow \lambda e \cdot c_1 \rangle\!\!\rangle \left(\phi_2 \supset \vec{\phi}\right)(f) \end{array}$$

For *elim*, we need to show:

$$\forall \phi_2 \in \vec{\phi} \left(\left(\phi_1 \supset \vec{\phi}\right) \wedge \phi_1\right) \overset{e_{\mathsf{eval}}}{\to} \phi_2$$

For this, we first prove the following uncurrying lemma, in which we use the abbreviation $\rho\, e := \left\langle \lambda^0.e \bullet (0 \bullet e_{\mathsf{p}_1}) \bullet (0 \bullet e_{\mathsf{p}_2}) \right\rangle$.

▶ **Lemma D14** (Universal Implication Uncurrying)**.**

$$\frac{\phi_1 \overset{e}{\to} \left(\phi_2 \supset \vec{\phi}\right)}{\forall \phi_3 \in \vec{\phi}.\, (\phi_1 \wedge \phi_2) \overset{\rho\, e}{\to} \phi_3}$$

Proof. We need to prove:

$$\frac{\forall c \in \mathbb{A}. \phi_1\,(c) \leq \langle\!\!\langle r \leftarrow e \cdot c \rangle\!\!\rangle \left(\phi_2 \supset \vec{\phi}\right)(r)}{\forall \phi_3 \in \vec{\phi}. \forall c \in \mathbb{A}.\, (\phi_1 \wedge \phi_2)\,(c) \leq \langle\!\!\langle r \leftarrow \rho\, e \cdot c \rangle\!\!\rangle\, \phi_3\,(r)}$$

Given a $\phi_3 \in \vec{\phi}$, due to the lower bound property of $\bigsqcap$, we have:

$$\begin{array}{ll} & \bigsqcap_{\phi_3 \in \vec{\phi}} \bigsqcap_{c_2 \in \mathbb{A}} \left(\phi_2\,(c_2) \sqsupset \langle\!\!\langle r \leftarrow f \cdot c_2 \rangle\!\!\rangle\, \phi_3\,(r)\right) \\ \leq & \bigsqcap_{c_2 \in \mathbb{A}} \left(\phi_2\,(c_2) \sqsupset \langle\!\!\langle r \leftarrow f \cdot c_2 \rangle\!\!\rangle\, \phi_3\,(r)\right) \\ \underset{(13)}{\leq} & \langle\!\!\langle c_2 \leftarrow c \cdot e_{\mathsf{p}_2} \rangle\!\!\rangle\, \phi_2\,(c_2) \sqsupset \langle\!\!\langle c_2 \leftarrow c \cdot e_{\mathsf{p}_2} \rangle\!\!\rangle \langle\!\!\langle r \leftarrow f \cdot c_2 \rangle\!\!\rangle\, \phi_3\,(r) \end{array}$$

By Lem. D12, we get that for any $c_1 \in \mathbb{A}$:

$$\begin{array}{rl} & \langle\!\rangle f \leftarrow e \cdot c_1 \rangle\!\rangle \bigsqcap_{\phi_3 \in \vec{\phi}} \bigsqcap_{c_2 \in \mathbb{A}} \left(\phi_2\left(c_2\right) \sqsupset \langle\!\rangle r \leftarrow f \cdot c_2 \rangle\!\rangle \phi_3\left(r\right)\right) \\ \leq & \langle\!\rangle f \leftarrow e \cdot c_1 \rangle\!\rangle \langle\!\rangle c_2 \leftarrow c \cdot e_{\mathsf{p}_2} \rangle\!\rangle \phi_2\left(c_2\right) \sqsupset \langle\!\rangle c_2 \leftarrow c \cdot e_{\mathsf{p}_2} \rangle\!\rangle \langle\!\rangle r \leftarrow f \cdot c_2 \rangle\!\rangle \phi_3\left(r\right) \end{array}$$

Instantiating our premise with c_1 obtains:

$$\begin{array}{rl} & \phi_1\left(c_1\right) \\ \leq & \langle\!\rangle f \leftarrow e \cdot c_1 \rangle\!\rangle \bigsqcap_{\phi_3 \in \vec{\phi}} \bigsqcap_{c_2 \in \mathbb{A}} \left(\phi_2\left(c_2\right) \sqsupset \langle\!\rangle r \leftarrow f \cdot c_2 \rangle\!\rangle \phi_3\left(r\right)\right) \\ \underset{Lem.\ D12}{\leq} & \langle\!\rangle f \leftarrow e \cdot c_1 \rangle\!\rangle \left(\langle\!\rangle c_2 \leftarrow c \cdot e_{\mathsf{p}_2} \rangle\!\rangle \phi_2\left(c_2\right) \sqsupset \langle\!\rangle c_2 \leftarrow c \cdot e_{\mathsf{p}_2} \rangle\!\rangle \langle\!\rangle r \leftarrow f \cdot c_2 \rangle\!\rangle \phi_3\left(r\right)\right) \\ \underset{Lem.\ D13}{\leq} & \langle\!\rangle c_2 \leftarrow c \cdot e_{\mathsf{p}_2} \rangle\!\rangle \phi_2\left(c_2\right) \sqsupset \langle\!\rangle f \leftarrow e \cdot c_1 \rangle\!\rangle \langle\!\rangle c_2 \leftarrow c \cdot e_{\mathsf{p}_2} \rangle\!\rangle \langle\!\rangle r \leftarrow f \cdot c_2 \rangle\!\rangle \phi_3\left(r\right) \end{array}$$

So by uncurrying $\langle\!\rangle c_2 \leftarrow c \cdot e_{\mathsf{p}_2} \rangle\!\rangle \phi_2\left(c_2\right)$ and then currying $\phi_1\left(c_1\right)$ we get:

$$\begin{array}{rl} & \langle\!\rangle c_2 \leftarrow c \cdot e_{\mathsf{p}_2} \rangle\!\rangle \phi_2\left(c_2\right) \\ \leq & \phi_1\left(c_1\right) \sqsupset \langle\!\rangle f \leftarrow e \cdot c_1 \rangle\!\rangle \langle\!\rangle c_2 \leftarrow c \cdot e_{\mathsf{p}_2} \rangle\!\rangle \langle\!\rangle r \leftarrow f \cdot c_2 \rangle\!\rangle \phi_3\left(r\right) \end{array}$$

Now, for every $c_1 \in \mathbb{A}$, we have that:

$$\begin{array}{rl} & \langle\!\rangle c_2 \leftarrow c \cdot e_{\mathsf{p}_2} \rangle\!\rangle \phi_2\left(c_2\right) \\ \leq & \bigsqcap_{c_1} \left(\phi_1\left(c_1\right) \sqsupset \langle\!\rangle f \leftarrow e \cdot c_1 \rangle\!\rangle \langle\!\rangle c_2 \leftarrow c \cdot e_{\mathsf{p}_2} \rangle\!\rangle \langle\!\rangle r \leftarrow f \cdot c_2 \rangle\!\rangle \phi_3\left(r\right)\right) \\ \underset{(13)}{\leq} & \langle\!\rangle c_1 \leftarrow c \cdot e_{\mathsf{p}_1} \rangle\!\rangle \phi_1\left(c_1\right) \sqsupset \langle\!\rangle c_1 \leftarrow c \cdot e_{\mathsf{p}_1} \rangle\!\rangle \langle\!\rangle f \leftarrow e \cdot c_1 \rangle\!\rangle \langle\!\rangle c_2 \leftarrow c \cdot e_{\mathsf{p}_2} \rangle\!\rangle \langle\!\rangle r \leftarrow f \cdot c_2 \rangle\!\rangle \phi_3\left(r\right) \end{array}$$

Therefore, by uncurrying again we get:

$$\begin{array}{rl} & \left(\phi_1 \wedge \phi_2\right)\left(c\right) \\ = & \langle\!\rangle c_1 \leftarrow c \cdot e_{\mathsf{p}_1} \rangle\!\rangle \phi_1\left(c_1\right) \sqcap \langle\!\rangle c_2 \leftarrow c \cdot e_{\mathsf{p}_2} \rangle\!\rangle \phi_2\left(c_2\right) \\ \leq & \langle\!\rangle c_1 \leftarrow c \cdot e_{\mathsf{p}_1} \rangle\!\rangle \langle\!\rangle f \leftarrow e \cdot c_1 \rangle\!\rangle \langle\!\rangle c_2 \leftarrow c \cdot e_{\mathsf{p}_2} \rangle\!\rangle \langle\!\rangle r \leftarrow f \cdot c_2 \rangle\!\rangle \phi_3\left(r\right) \\ \underset{\text{A.Bind}}{\leq} & \langle\!\rangle r \leftarrow (\mathsf{let}\ c_1 \ \Leftarrow c \cdot e_{\mathsf{p}_1}\ \mathsf{in}\ \mathsf{let}\ f \ \Leftarrow e \cdot c_1 \mathsf{let}\ c_2 \ \Leftarrow c \cdot e_{\mathsf{p}_2}\ \mathsf{in}\ f \cdot c_2\ \mathsf{in}\ ) \rangle\!\rangle \phi_3\left(r\right) \\ = & \langle\!\rangle r \leftarrow \nu\left(e \bullet \left(c \bullet e_{\mathsf{p}_1}\right) \bullet \left(c \bullet e_{\mathsf{p}_2}\right)\right) \rangle\!\rangle \phi_3\left(r\right) \\ = & \langle\!\rangle r \leftarrow \left\langle \lambda^0.e \bullet \left(0 \bullet e_{\mathsf{p}_1}\right) \bullet \left(0 \bullet e_{\mathsf{p}_2}\right)\right\rangle \cdot c \rangle\!\rangle \phi_3\left(r\right) \\ = & \langle\!\rangle r \leftarrow \rho\, e \cdot c \rangle\!\rangle \phi_3\left(r\right) \end{array}$$

◀

To get back to the proof of *elim*, by Reflexivity, we have:

$$\phi_1 \supset \vec{\phi} \overset{e_{\mathsf{id}}}{\to} \phi_1 \supset \vec{\phi}$$

From which, by Lem. D14, we get:

$$\forall \phi_2 \in \vec{\phi}.\ \left(\left(\phi_1 \supset \vec{\phi}\right) \wedge \phi_1\right) \overset{\rho\, e_{\mathsf{id}}}{\to} \phi_2$$

From this, we get the desired result, since $e_{\mathsf{eval}} = \rho\, e_{\mathsf{id}}$. ◀

▶ **Theorem 18** (🦜). *Let $\mathcal{MC}_M = (\mathbb{A}, \Omega, \Diamond, \mathcal{S})$ be a monadic core. Then the induced evidenced frame has an evidence $e \in \mathbb{A}$ such that $\top \overset{e}{\to} \bot$ iff $\mathbf{1} \leq \mathbf{0}$.*

Proof. ($\Rightarrow$) If there exists $e \in \mathcal{S}$ such that $\top \overset{e}{\to} \bot$, then for every $c \in \mathbb{A}$:

$$\mathbf{1} = \top\left(c\right) \leq \langle\!\rangle r \leftarrow e \cdot c \rangle\!\rangle \bot\left(r\right) = \langle\!\rangle r \leftarrow e \cdot c \rangle\!\rangle \mathbf{0} \leq \mathbf{0}$$

($\Leftarrow$) If $\mathbf{1} \leq \mathbf{0}$, then $\top \overset{e_{\mathsf{id}}}{\to} \bot$ since for every $c \in \mathbb{A}$:

$$\top\left(c\right) = \mathbf{1} \leq \mathbf{0} \underset{\text{A.Return}}{\leq} \langle\!\rangle r \leftarrow [c] \rangle\!\rangle \mathbf{0} = \langle\!\rangle r \leftarrow e_{\mathsf{id}} \cdot c \rangle\!\rangle \bot\left(r\right)$$ ◀