GALOIS CONNECTING CALL-BY-VALUE AND CALL-BY-NAME

DYLAN MCDERMOTT [a] AND ALAN MYCROFT [b]

[a] Reykjavik University, Iceland
e-mail address: dylanm@ru.is

[b] University of Cambridge, UK
e-mail address: Alan.Mycroft@cl.cam.ac.uk

Abstract. We establish a general framework for reasoning about the relationship between call-by-value and call-by-name.

In languages with side-effects, call-by-value and call-by-name executions of programs often have different, but related, observable behaviours. For example, if a program might diverge but otherwise has no side-effects, then whenever it terminates under call-by-value, it terminates with the same result under call-by-name. We propose a technique for stating and proving these properties. The key ingredient is Levy's call-by-push-value calculus, which we use as a framework for reasoning about evaluation orders. We show that the call-by-value and call-by-name translations of expressions into call-by-push-value have related observable behaviour under certain conditions on side-effects, which we identify. We then use this fact to construct maps between the call-by-value and call-by-name interpretations of types, and identify further properties of side-effects that imply these maps form a Galois connection. These properties hold for some side-effects (such as divergence), but not others (such as mutable state). This gives rise to a general reasoning principle that relates call-by-value and call-by-name. We apply the reasoning principle to example side-effects including divergence and nondeterminism.

1. Introduction

Suppose that we have a language in which terms can be statically tagged either as using call-by-value evaluation or as using call-by-name evaluation. Each program in this language would therefore use a mix of call-by-value and call-by-name at runtime. Given any such program M, we can construct a new program M' by changing call-by-value to call-by-name for some subterm. The question we consider in this paper is: what is the relationship between the observable behaviour of M and the observable behaviour of M'?

For a language with side-effects (such as divergence), changing the evaluation order in this way will in general change the behaviour of the program, but for some side-effects we can often say something about how we expect the behaviour to change:

- If there are no side-effects at all (in particular, programs are normalizing), the choice of evaluation order is irrelevant: M and M' terminate with the same result.

Key words and phrases: computational effect, evaluation order, call-by-push-value, categorical semantics.
* This article is an extended version of [MM22].

Preprint submitted to
Logical Methods in Computer Science

© GALOIS CONNECTING CALL-BY-VALUE AND CALL-BY-NAME
Creative Commons

- If there are diverging terms (for instance, via recursion), then the behaviour may change: a program might diverge under call-by-value and return a result under call-by-name. However, we can say something about how the behaviour changes: if M terminates with some result, then M' terminates with the same result.
- If nondeterminism is the only side-effect, every result of M is a possible result of M'.

These three instances of the problem are intuitively obvious, and each can be proved separately. We develop a *general* technique for proving these properties.

The idea is to use a calculus that captures both call-by-value and call-by-name, as a setting in which we can reason about both evaluation orders (this is where M and M' live). The calculus we use is Levy's *call-by-push-value* (CBPV) [Lev99]. Levy describes how to translate (possibly open) expressions e into CBPV terms $\mathcal{V}(\!|e|\!)$ and $\mathcal{N}(\!|e|\!)$, which respectively correspond to call-by-value and call-by-name. We study the relationship between the behaviour of $\mathcal{V}(\!|e|\!)$ and the behaviour of $\mathcal{N}(\!|e|\!)$ in a given program context.

The main obstacle is that $\mathcal{V}(\!|e|\!)$ and $\mathcal{N}(\!|e|\!)$ have different types. (The former has a "call-by-value type" $\mathbf{F}\,(\mathcal{V}(\!|\tau|\!))$ and the latter a "call-by-name type" $\mathcal{N}(\!|\tau|\!)$, defined in Section 2.1.) They hence cannot be directly compared. Our solution is based on Reynolds's work relating direct and continuation semantics of the λ-calculus [Rey74].

The first step is to define a family of (set-theoretic) relations (in the style of a logical relation) that compares the observable behaviour of a term of call-by-value type with observable behaviour of a term of call-by-name type. We can then ask whether $\mathcal{V}(\!|e|\!)$ is related in this sense to $\mathcal{N}(\!|e|\!)$. This is not the case in general. In the presence of arbitrary side-effects, we cannot expect to say anything useful about how the behaviour of $\mathcal{V}(\!|e|\!)$ relates to the behaviour of $\mathcal{N}(\!|e|\!)$. However, under certain conditions satisfied only for certain side-effects, $\mathcal{V}(\!|e|\!)$ is related to $\mathcal{N}(\!|e|\!)$. These conditions say roughly that we can discard, duplicate, and reorder side-effects. The main result of the first step is a theorem relating the two translations of e when these conditions hold (Theorem 4.7). This does not quite say what happens if we were to replace call-by-value with call-by-name within some program; that is the goal of the second step.

The second step is to identify maps between the call-by-value and call-by-name interpretations, forming *Galois connections* (one for each source-language type) between the two interpretations. We compose these maps with the translations of expressions, to arrive at two terms that *can* be compared directly. For this step we assume a stronger condition on side-effects than in the first, saying informally that we can thunk side-effects. Under this condition we show that the maps between call-by-value and call-by-name *represent* the relations from the first step. By combining this fact with Theorem 4.7 we prove a result that directly relates the two terms we construct by composition with the Galois connections.

We therefore arrive at a general *reasoning principle* (Theorem 6.2) that we use to compare call-by-value with call-by-name. Given any preorder $\preccurlyeq$ that captures the property we wish to show about programs, our reasoning principle gives conditions that imply $M \preccurlyeq M'$, where M' is constructed as above by replacing call-by-value with call-by-name. We apply our reasoning principle to examples by choosing different relations $\preccurlyeq$; each of these relations indicates the extent to which changing evaluation order affects the behaviour of the program. In the divergence example $N \preccurlyeq N'$ is defined to mean termination of N implies termination of N' with the same result; in the other examples $\preccurlyeq$ similarly mirrors the properties described informally above.

Rather than just considering some fixed collection of (allowable) side-effects, we work abstractly and identify properties of side-effects that enable us to relate call-by-value and call-by-name.

Our reasoning principle relies on the existence of some denotational model of the side-effects. We construct the Galois connections and relate the call-by-value and call-by-name translations inside the model itself. Crucially, we use *order-enriched* models, which order the denotations of terms. The ordering on denotations is necessary to obtain a general reasoning principle. (Our example properties cannot be proved by showing that denotations are equal, because they are not symmetric.) Working inside the semantics rather than using syntactic logical relations makes it easier to prove and to use our reasoning principle, especially for the divergence example.

In Section 2 we summarize the call-by-push-value calculus (CBPV) and the call-by-value and call-by-name translations. We then make the following contributions:

- We describe an *order-enriched* categorical semantics for CBPV (Section 3).
- We define a family of relations for comparing the observable behaviours of a term of call-by-value type with a term of call-by-name type (Section 4). We prove that, for side-effects satisfying certain conditions, the call-by-value and call-by-name translations of expressions are related by these (Theorem 4.7). As a corollary, we directly relate the call-by-value and call-by-name translations of closed expressions of type **bool** (Corollary 4.8).
- We define the Galois connections between the call-by-value and call-by-name translations (Section 5), and show that they represent the relations from the first step (Lemma 5.8).
- We use the Galois connections to prove a novel reasoning principle (Theorem 6.2) that relates the call-by-value and call-by-name translations of expressions (Section 6).

We apply our reasoning principle to three different examples: no side-effects, divergence, and nondeterminism. In this way we establish all of three facts listed at the beginning of this introduction. Our motivation is partly to demonstrate the Galois connection technique as a way of reasoning about different semantics of a given language. Call-by-value and call-by-name is one example of this (and Reynolds's original application to direct and continuation semantics is another).

This paper is a revised and extended version of [MM22]. The primary difference is the addition of Section 4, containing the first step outlined above. (The conference version [MM22] skips this step and goes directly to the Galois connections.) This in particular enables us to prove a statement about closed terms of type **bool** (Corollary 4.8) under weaker assumptions than in the conference version [MM22, Corollary 22]. We also add an extra example (immutable state), add products to the source language, and include more detailed proofs than in the conference version.

2. Call-by-push-value, call-by-value, and call-by-name

Levy [Lev99, Lev06] introduced call-by-push-value (CBPV) as a calculus that captures both call-by-value and call-by-name. We reason about the relationship between call-by-value and call-by-name evaluation inside CBPV.

The syntax of CBPV terms is stratified into two kinds: *values* V, W do not reduce, *computations* M, N might reduce (possibly with side-effects). The syntax of types is similarly

stratified into *value types* A, B and *computation types* $\underline{C}, \underline{D}$.

$$
\begin{array}{rrcl}
\text{value types} & A, B & ::= & \mathbf{unit} \mid A_1 \times A_2 \mid \mathbf{bool} \mid \mathbf{U}\,\underline{C} \\
\text{computation types} & \underline{C}, \underline{D} & ::= & \underline{C}_1 \mathbin{\&} \underline{C}_2 \mid A \to \underline{C} \mid \mathbf{F}\,A \\
\text{values} & V, W & ::= & x \mid () \mid (V_1, V_2) \mid \mathbf{true} \mid \mathbf{false} \mid \mathbf{thunk}\,M \\
\text{computations} & M, N & ::= & \lambda\{1.\,M_1, 2.\,M_2\} \mid 1\text{`}M \mid 2\text{`}M \\
& & & \mid \lambda x{:}A.\,M \mid V\text{`}M \mid \mathbf{return}\,V \mid M\ \mathbf{to}\ x.\,N \\
& & & \mid \mathbf{match}\ V\ \mathbf{with}\ (x_1, x_2).\,M \\
& & & \mid \mathbf{if}\ V\ \mathbf{then}\ M_1\ \mathbf{else}\ M_2 \mid \mathbf{force}\,V
\end{array}
$$

We restrict to only the subset of CBPV required for this paper.

The value type $\mathbf{U}\,\underline{C}$ is the type of *thunks* of computations of type $\underline{C}$. Elements of $\mathbf{U}\,\underline{C}$ are introduced using **thunk**: the value $\mathbf{thunk}\,M$ is the suspension of the computation term M. The corresponding eliminator is **force**, which is the inverse of **thunk**. Computation types include binary products; the pairing of two computations M_1 and M_2 is written $\lambda\{1.\,M_1, 2.\,M_2\}$, and the first and second projections are $1\text{`}M$ and $2\text{`}M$. Computation types also include function types (where functions send values to computations). Function application is written $V\text{`}M$, where V is the argument and M is the function to apply. The *returner type* $\mathbf{F}\,A$ has as elements computations that return elements of the value type A; these computations may have side-effects. Elements of $\mathbf{F}\,A$ are introduced by **return**; the computation $\mathbf{return}\,V$ immediately returns the value V (with no side-effects). Computations can be sequenced using $M\ \mathbf{to}\ x.\,N$. This first evaluates M (which is required to have returner type), and then evaluates N with x bound to the result of M. (It is similar to `M >>= \x -> N` in Haskell.) The syntax we give here does not include any method of introducing effects; we extend CBPV with some example side-effects in Section 2.2.

The evaluation order in CBPV is fixed for each program. The only primitive that causes the evaluation of two separate computations is **to**, which implements eager sequencing. Thunks give us more control over the evaluation order: they can be arbitrarily duplicated and discarded, and can be forced in any order chosen by the program. This is how CBPV captures both call-by-value and call-by-name.

CBPV has two typing judgments: $\Gamma \vdash V : A$ for values and $\Gamma \vdash_c M : \underline{C}$ for computations. *Typing contexts* Γ are ordered lists of (variable, value type) pairs. We require that no variable appears more than once in any typing context. Figure 1 gives the typing rules. Rules that add a new variable to a typing context implicitly require that the variable is fresh. We write $\diamond$ for the empty typing context, $V : A$ as an abbreviation for $\diamond \vdash V : A$, and $M : \underline{C}$ as an abbreviation for $\diamond \vdash_c M : \underline{C}$.

We give an operational semantics for CBPV. This consists of a big-step evaluation relation $M \Downarrow R$, which means the computation M evaluates to R. Here R ranges over *terminal computations*, which are the subset of computations with an introduction form on the outside:

$$R \ ::= \ \lambda\{1.\,M_1, 2.\,M_2\} \mid \lambda x{:}A.\,M \mid \mathbf{return}\,V$$

We only evaluate closed, well-typed computations, so when we write $M \Downarrow R$ we assume $M : \underline{C}$ for some $\underline{C}$ (this implies $R : \underline{C}$). Reduction therefore cannot get stuck. The rules defining $\Downarrow$ are given in Figure 2. All terminal computations evaluate to themselves. Products of computations are *lazy*: to evaluate a projection $i\text{`}M$, only the ith component of the pair M is evaluated. Since we have not yet included any way of forming impure computations, the

$$\boxed{\Gamma \vdash V : A}$$

$$\frac{}{\Gamma \vdash x : A}\ \text{if } (x : A) \in \Gamma \qquad \frac{}{\Gamma \vdash () : \mathbf{unit}} \qquad \frac{\Gamma \vdash V_1 : A_1 \qquad \Gamma \vdash V_2 : A_2}{\Gamma \vdash (V_1, V_2) : A_1 \times A_2}$$

$$\frac{}{\Gamma \vdash \mathbf{true} : \mathbf{bool}} \qquad \frac{}{\Gamma \vdash \mathbf{false} : \mathbf{bool}} \qquad \frac{\Gamma \vdash_c M : \underline{C}}{\Gamma \vdash \mathbf{thunk}\, M : \mathbf{U}\,\underline{C}}$$

$$\boxed{\Gamma \vdash_c M : \underline{C}}$$

$$\frac{\Gamma \vdash_c M_1 : \underline{C}_1 \qquad \Gamma \vdash_c M_2 : \underline{C}_2}{\Gamma \vdash_c \lambda\{1.\, M_1, 2.\, M_2\} : \underline{C}_1 \mathbin{\&} \underline{C}_2} \qquad \frac{\Gamma \vdash_c M : \underline{C}_1 \mathbin{\&} \underline{C}_2}{\Gamma \vdash_c 1\text{`}M : \underline{C}_1} \qquad \frac{\Gamma \vdash_c M : \underline{C}_1 \mathbin{\&} \underline{C}_2}{\Gamma \vdash_c 2\text{`}M : \underline{C}_2}$$

$$\frac{\Gamma, x : A \vdash_c M : \underline{C}}{\Gamma \vdash_c \lambda x{:}A.\, M : A \to \underline{C}} \qquad \frac{\Gamma \vdash V : A \qquad \Gamma \vdash_c M : A \to \underline{C}}{\Gamma \vdash_c V\text{`}M : \underline{C}} \qquad \frac{\Gamma \vdash V : A}{\Gamma \vdash_c \mathbf{return}\, V : \mathbf{F}\, A}$$

$$\frac{\Gamma \vdash_c M : \mathbf{F}\, A \qquad \Gamma, x : A \vdash_c N : \underline{C}}{\Gamma \vdash_c M\ \mathbf{to}\ x.\, N : \underline{C}} \qquad \frac{\Gamma \vdash V : A_1 \times A_2 \qquad \Gamma, x_1 : A_1, x_2 : A_2 \vdash_c M : \underline{C}}{\Gamma \vdash_c \mathbf{match}\ V\ \mathbf{with}\ (x_1, x_2).\, M : \underline{C}}$$

$$\frac{\Gamma \vdash V : \mathbf{bool} \qquad \Gamma \vdash_c M_1 : \underline{C} \qquad \Gamma \vdash_c M_2 : \underline{C}}{\Gamma \vdash_c \mathbf{if}\ V\ \mathbf{then}\ M_1\ \mathbf{else}\ M_2 : \underline{C}} \qquad \frac{\Gamma \vdash V : \mathbf{U}\,\underline{C}}{\Gamma \vdash_c \mathbf{force}\, V : \underline{C}}$$

FIGURE 1. CBPV typing rules

$$\frac{}{\lambda\{1.\, M_1, 2.\, M_2\} \Downarrow \lambda\{1.\, M_1, 2.\, M_2\}} \qquad \frac{M \Downarrow \lambda\{1.\, N_1, 2.\, N_2\} \qquad N_i \Downarrow R}{i\text{`}M \Downarrow R}\ i \in \{1, 2\}$$

$$\frac{}{\lambda x{:}A.\, M \Downarrow \lambda x{:}A.\, M} \qquad \frac{M \Downarrow \lambda x{:}A.\, N \qquad N[x \mapsto V] \Downarrow R}{V\text{`}M \Downarrow R}$$

$$\frac{}{\mathbf{return}\, V \Downarrow \mathbf{return}\, V} \qquad \frac{M \Downarrow \mathbf{return}\, V \qquad N[x \mapsto V] \Downarrow R}{M\ \mathbf{to}\ x.\, N \Downarrow R}$$

$$\frac{M_1 \Downarrow R}{\mathbf{if\ true\ then}\ M_1\ \mathbf{else}\ M_2 \Downarrow R} \qquad \frac{M_2 \Downarrow R}{\mathbf{if\ false\ then}\ M_1\ \mathbf{else}\ M_2 \Downarrow R}$$

$$\frac{M[x_1 \mapsto V_1, x_2 \mapsto V_2] \Downarrow R}{\mathbf{match}\ (V_1, V_2)\ \mathbf{with}\ (x_1, x_2).\, M \Downarrow R} \qquad \frac{M \Downarrow R}{\mathbf{force}\,(\mathbf{thunk}\, M) \Downarrow R}$$

FIGURE 2. Big-step operational semantics of CBPV

semantics is deterministic and normalizing: given any $M : \underline{C}$, there is exactly one terminal computation R such that $M \Downarrow R$. Section 2.2 extends the semantics in ways that violate these properties. We are primarily interested in evaluating computations of returner type.

A CBPV *program* is a closed computation $M : \mathbf{F}\,\mathbf{bool}$. The reasoning principle we give for call-by-value and call-by-name relates open terms in program contexts. A *program relation* consists of a preorder[1] $\preccurlyeq$ on programs. For example, we could use

$$M \preccurlyeq M' \quad \text{if and only if} \quad \forall V : \mathbf{bool}.\ (M \Downarrow \mathbf{return}\, V) \ \Rightarrow\ (M' \Downarrow \mathbf{return}\, V)$$

We could also use, for example, the total relation for $\preccurlyeq$ (and in this case apply our reasoning principle for call-by-value and call-by-name even if we include e.g. mutable state as a side effect – but then of course the conclusion of our reasoning principle would be trivial). Given any program relation $\preccurlyeq$, we define a *contextual preorder* $M \preccurlyeq^{\Gamma}_{\mathrm{ctx}} M'$ on arbitrary well-typed computations (in typing context Γ) by considering the behaviour of M and M' in programs as follows. A *computation context* $\mathcal{E}$ is a computation term, with a single hole $\Box$ where a computation term is expected. We write $\mathcal{E}[M]$ for the computation that results from replacing $\Box$ with M (which may capture some of the free variables of M). For example, if $\mathcal{E}$ is the computation context N **to** $x.\,\Box$ then $\mathcal{E}[\mathbf{return}\, x]$ is the computation N **to** $x.\,\mathbf{return}\, x$, where x is captured. We use computation contexts to define $\preccurlyeq^{\Gamma}_{\mathrm{ctx}}$.

Definition 2.1 (Contextual preorder)**.** Suppose that $\preccurlyeq$ is a program relation, and that $\Gamma \vdash_c M : \underline{C}$ and $\Gamma \vdash_c M' : \underline{C}$ are two computations of the same type. We write $M \preccurlyeq^{\Gamma}_{\mathrm{ctx}} M'$ if, for all computation contexts $\mathcal{E}$ such that $\mathcal{E}[M], \mathcal{E}[M'] : \mathbf{F}\,\mathbf{bool}$, we have $\mathcal{E}[M] \preccurlyeq \mathcal{E}[M']$. We write $M \cong^{\Gamma}_{\mathrm{ctx}} M'$, and say that M and M' are *contextually equivalent*, when both $M \preccurlyeq^{\Gamma}_{\mathrm{ctx}} M'$ and $M' \preccurlyeq^{\Gamma}_{\mathrm{ctx}} M$ hold.

We sometimes omit Γ, and write just $M \preccurlyeq_{\mathrm{ctx}} M'$ or $M \cong_{\mathrm{ctx}} M'$.

2.1. **Call-by-value and call-by-name.** We use CBPV (instead of e.g. Moggi's monadic metalanguage [Mog91]) because it captures both call-by-value and call-by-name evaluation. Levy [Lev99] gives two compositional translations from a source language into CBPV: one for call-by-value and one for call-by-name. We recall both translations in this section; our goal is to reason about the relationship between them.

For the source language, we use the following syntax of types τ and expressions e:

$$\tau \ ::=\ \mathbf{unit} \mid \tau_1 \times \tau_2 \mid \tau \to \tau'$$

$$e \ ::=\ x \mid () \mid (e_1, e_2) \mid \mathbf{fst}\, e \mid \mathbf{snd}\, e \mid \mathbf{true} \mid \mathbf{false} \mid \mathbf{if}\ e_0\ \mathbf{then}\ e_1\ \mathbf{else}\ e_2 \mid \lambda x{:}\tau.\, e \mid e\, e'$$

The source language has a typing judgement of the form $\Gamma \vdash e : \tau$, defined by the usual rules.

The two translations from the source language to CBPV are defined in Figure 3. For call-by-value, each source language type τ is mapped to a CBPV value type $\mathcal{V}(\!|\tau|\!)$ that contains the results of call-by-value computations. For call-by-name, τ is translated to a computation type $\mathcal{N}(\!|\tau|\!)$, which contains the computations themselves. Products in call-by-value use the value-type products of CBPV (which means they are necessarily *strict*: both components of a pair are always evaluated). For call-by-name we give a lazy interpretation of binary products, using products of CBPV computation types. (Though note that we do not interpret **unit** as a nullary product of computation types.) Functions under the call-by-value translation accept values of type $\mathcal{V}(\!|\tau|\!)$ as arguments; arguments are evaluated before being passed to the function. Under the call-by-name translation, functions accept thunks of computations

[1]We do not actually need to assume that $\preccurlyeq$ is reflexive or transitive at any point, but because of constraints we add later (such as existence of an adequate model), we do not expect there to be any interesting examples in which $\preccurlyeq$ is not a preorder.

$$\boxed{\text{type } \tau \;\mapsto\; \text{value type } \mathcal{V}(\!|\tau|\!)}$$

$$\begin{array}{rcl}
\mathbf{unit} & \mapsto & \mathbf{unit} \\
\tau_1 \times \tau_2 & \mapsto & \mathcal{V}(\!|\tau_1|\!) \times \mathcal{V}(\!|\tau_2|\!) \\
\mathbf{bool} & \mapsto & \mathbf{bool} \\
\tau \to \tau' & \mapsto & \mathbf{U}\,(\mathcal{V}(\!|\tau|\!) \to \mathbf{F}\,(\mathcal{V}(\!|\tau'|\!)))
\end{array}$$

$$\boxed{\text{typing context } \Gamma \;\mapsto\; \text{typing context } \mathcal{V}(\!|\Gamma|\!)}$$

$$\begin{array}{rcl}
\diamond & \mapsto & \diamond \\
\Gamma, x : \tau & \mapsto & \mathcal{V}(\!|\Gamma|\!), x : \mathcal{V}(\!|\tau|\!)
\end{array}$$

$$\boxed{\text{expression } \Gamma \vdash e : \tau \;\mapsto\; \text{computation } \mathcal{V}(\!|\Gamma|\!) \vdash_c \mathcal{V}(\!|e|\!) : \mathbf{F}\,(\mathcal{V}(\!|\tau|\!))}$$

$$\begin{array}{rcl}
x & \mapsto & \mathbf{return}\, x \\
() & \mapsto & \mathbf{return}\, () \\
(e_1, e_2) & \mapsto & \mathcal{V}(\!|e_1|\!)\ \mathbf{to}\ z_1.\, \mathcal{V}(\!|e_2|\!)\ \mathbf{to}\ z_2.\, \mathbf{return}\, (z_1, z_2) \\
\mathbf{fst}\, e & \mapsto & \mathcal{V}(\!|e|\!)\ \mathbf{to}\ z.\, \mathbf{match}\ z\ \mathbf{with}\ (z_1, z_2).\, \mathbf{return}\, z_1 \\
\mathbf{snd}\, e & \mapsto & \mathcal{V}(\!|e|\!)\ \mathbf{to}\ z.\, \mathbf{match}\ z\ \mathbf{with}\ (z_1, z_2).\, \mathbf{return}\, z_2 \\
\mathbf{true} & \mapsto & \mathbf{return}\, \mathbf{true} \\
\mathbf{false} & \mapsto & \mathbf{return}\, \mathbf{false} \\
\mathbf{if}\ e_0\ \mathbf{then}\ e_1\ \mathbf{else}\ e_2 & \mapsto & \mathcal{V}(\!|e_0|\!)\ \mathbf{to}\ z.\, \mathbf{if}\ z\ \mathbf{then}\ \mathcal{V}(\!|e_1|\!)\ \mathbf{else}\ \mathcal{V}(\!|e_2|\!) \\
\lambda x{:}\tau.\, e & \mapsto & \mathbf{return}\, \mathbf{thunk}\, \lambda x{:}\mathcal{V}(\!|\tau|\!).\, \mathcal{V}(\!|e|\!) \\
e\, e' & \mapsto & \mathcal{V}(\!|e|\!)\ \mathbf{to}\ y.\, \mathcal{V}(\!|e'|\!)\ \mathbf{to}\ z.\, z\, `\, \mathbf{force}\, y
\end{array}$$

(A) Call-by-value translation $\mathcal{V}(\!|-|\!)$

$$\boxed{\text{type } \tau \;\mapsto\; \text{computation type } \mathcal{N}(\!|\tau|\!)}$$

$$\begin{array}{rcl}
\mathbf{unit} & \mapsto & \mathbf{F}\ \mathbf{unit} \\
\tau_1 \times \tau_2 & \mapsto & \mathcal{N}(\!|\tau_1|\!) \times \mathcal{N}(\!|\tau_2|\!) \\
\mathbf{bool} & \mapsto & \mathbf{F}\ \mathbf{bool} \\
\tau \to \tau' & \mapsto & (\mathbf{U}\,(\mathcal{N}(\!|\tau|\!))) \to \mathcal{N}(\!|\tau'|\!)
\end{array}$$

$$\boxed{\text{typing context } \Gamma \;\mapsto\; \text{typing context } \mathcal{N}(\!|\Gamma|\!)}$$

$$\begin{array}{rcl}
\diamond & \mapsto & \diamond \\
\Gamma, x : \tau & \mapsto & \mathcal{N}(\!|\Gamma|\!), x : \mathbf{U}\,(\mathcal{N}(\!|\tau|\!))
\end{array}$$

$$\boxed{\text{expression } \Gamma \vdash e : \tau \;\mapsto\; \text{computation } \mathcal{N}(\!|\Gamma|\!) \vdash_c \mathcal{N}(\!|e|\!) : \mathcal{N}(\!|\tau|\!)}$$

$$\begin{array}{rcl}
x & \mapsto & \mathbf{force}\, x \\
() & \mapsto & \mathbf{return}\, () \\
(e_1, e_2) & \mapsto & \lambda\{1.\, \mathcal{N}(\!|e_1|\!), 2.\, \mathcal{N}(\!|e_2|\!)\} \\
\mathbf{fst}\, e & \mapsto & 1\, `\, \mathcal{N}(\!|e|\!) \\
\mathbf{snd}\, e & \mapsto & 2\, `\, \mathcal{N}(\!|e|\!) \\
\mathbf{true} & \mapsto & \mathbf{return}\, \mathbf{true} \\
\mathbf{false} & \mapsto & \mathbf{return}\, \mathbf{false} \\
\mathbf{if}\ e_0\ \mathbf{then}\ e_1\ \mathbf{else}\ e_2 & \mapsto & \mathcal{N}(\!|e_0|\!)\ \mathbf{to}\ z.\, \mathbf{if}\ z\ \mathbf{then}\ \mathcal{N}(\!|e_1|\!)\ \mathbf{else}\ \mathcal{N}(\!|e_2|\!) \\
\lambda x{:}\tau.\, e & \mapsto & \lambda x{:}\mathbf{U}\,(\mathcal{N}(\!|\tau|\!)).\, \mathcal{N}(\!|e|\!) \\
e\, e' & \mapsto & (\mathbf{thunk}\, \mathcal{N}(\!|e'|\!))\, `\, \mathcal{N}(\!|e|\!)
\end{array}$$

(B) Call-by-name translation $\mathcal{N}(\!|-|\!)$

FIGURE 3. Translations from the source language into CBPV

as arguments; instead of evaluating them, arguments are thunked before passing them to call-by-name functions. Source-language typing contexts Γ are translated to CBPV typing contexts $\mathcal{V}(\!|\Gamma|\!)$ and $\mathcal{N}(\!|\Gamma|\!)$. In call-by-value they contain values, in call-by-name they contain thunks of computations. Source-language expressions e are mapped to CBPV computations $\mathcal{V}(\!|e|\!)$ and $\mathcal{N}(\!|e|\!)$. The translation uses some auxiliary program variables, which are assumed fresh.

For call-by-value we arbitrarily choose left-to-right evaluation for both pairing and function application. Under the call-by-name translation, side-effects occur only at the base types **unit** and **bool** (since this is where the returner types appear).

Of course, we have to justify that these translations actually capture call-by-value and call-by-name. There are two semantics of interest for the source language: a call-by-value semantics (that evaluates left-to-right), and a call-by-name semantics (with lazy products). Since we consider the observable behaviour of CBPV terms, the properties we want are that if the call-by-value translations $\mathcal{V}(\!|e|\!)$ and $\mathcal{V}(\!|e'|\!)$ have the same observable behaviour then e and e' have the same observable behaviour with respect to the call-by-value semantics, and similarly for call-by-name. Levy [Lev99] proves both of these properties (though without products in the source language). We take this as the required justification, and do not give the details.

2.2. **Examples.** We consider three collections of (allowable) side-effects as examples throughout the paper.

Example 2.2 (No side-effects)**.** We include the simplest possible example: the case where there are no side-effects at all. For this example, call-by-value and call-by-name turn out to have identical behaviour. We define the program relation $M \preccurlyeq M'$ (for closed computations $M, M' : \mathbf{F}\,\mathbf{bool}$) as:

$$M \preccurlyeq M' \quad \text{if and only if} \quad \exists V : \mathbf{bool}.\ (M \Downarrow \mathbf{return}\, V)\ \wedge\ (M' \Downarrow \mathbf{return}\, V)$$

In other words, M and M' both evaluate to the same result V. Since evaluation is deterministic, V is necessarily unique. The contextual preorder $M \preccurlyeq^{\Gamma}_{\mathrm{ctx}} M'$ means if we construct two programs by wrapping M and M' in the same computation context, then these two programs evaluate to the same result. This relation is symmetric. Our other examples use non-symmetric relations.

Example 2.3 (Divergence)**.** For our second example, the only side-effect is divergence (via recursion). In this case, call-by-value and call-by-name do not have identical behaviour (they are not related by $\preccurlyeq_{\mathrm{ctx}}$ as it is defined in our no-side-effects example). We instead show that replacing call-by-value with call-by-name does not change a terminating program into a diverging one.

We extend our two languages with recursion. For CBPV we extend the syntax of computations with fixed points $\mathbf{rec}\, x : \mathbf{U}\,\underline{C}.\, M$, and correspondingly extend the type system and operational semantics with the following rules:

$$\frac{\Gamma, x : \mathbf{U}\,\underline{C} \vdash_c M : \underline{C}}{\Gamma \vdash_c \mathbf{rec}\, x : \mathbf{U}\,\underline{C}.\, M : \underline{C}} \qquad \frac{M[x \mapsto \mathbf{thunk}\,(\mathbf{rec}\, x : \mathbf{U}\,\underline{C}.\, M)] \Downarrow R}{\mathbf{rec}\, x : \mathbf{U}\,\underline{C}.\, M \Downarrow R}$$

The variable x is bound to a thunk of the recursive computation, so recursion is done by forcing x. (This is not the only way to add recursion to CBPV [DCL18], but is the most convenient for our purposes.) Of course, by adding recursion we lose normalization (but the

semantics is still deterministic). We extend the source language, and the two translations into CBPV, with recursive functions:

$$e \ ::= \ \ldots \mid \mathbf{rec}\, f : \tau \to \tau'.\, \lambda x.\, e \qquad \frac{\Gamma, f : \tau \to \tau', x : \tau \vdash e : \tau'}{\Gamma \vdash \mathbf{rec}\, f : \tau \to \tau'.\, \lambda x.\, e : \tau \to \tau'}$$

$$\begin{aligned} \mathcal{V}(\!|\mathbf{rec}\, f : \tau \to \tau'.\, \lambda x.\, e|\!) &= \mathbf{return}\, \mathbf{thunk}\, (\mathbf{rec}\, f : \mathbf{U}\, (\mathcal{V}(\!|\tau|\!) \to \mathbf{F}\, (\mathcal{V}(\!|\tau'|\!))).\, \lambda x : \mathcal{V}(\!|\tau|\!).\, \mathcal{V}(\!|e|\!)) \\ \mathcal{N}(\!|\mathbf{rec}\, f : \tau \to \tau'.\, \lambda x.\, e|\!) &= \mathbf{rec}\, f : \mathbf{U}\, (\mathbf{U}\, (\mathcal{N}(\!|\tau|\!)) \to \mathcal{N}(\!|\tau'|\!)).\, \lambda x : \mathbf{U}\, (\mathcal{N}(\!|\tau|\!)).\, \mathcal{N}(\!|e|\!) \end{aligned}$$

Again, the translations are the same as those given by Levy [Lev99], except that Levy has general fixed points for call-by-name, rather than just recursive functions. The expression $\Omega_\tau = ((\mathbf{rec}\, f : \mathbf{bool} \to \tau.\, \lambda x.\, f\, x)\, \mathbf{false}) : \tau$ enables us to distinguish between call-by-value and call-by-name: $(\lambda x : \tau.\, \mathbf{true})\, \Omega_\tau$ diverges in call-by-value but not in call-by-name. In particular, we have $\mathcal{N}(\!|(\lambda x : \tau.\, \mathbf{true})\, \Omega_\tau|\!) \Downarrow \mathbf{return}\, \mathbf{true}$, but there is no R such that $\mathcal{V}(\!|(\lambda x : \tau.\, \mathbf{true})\, \Omega_\tau|\!) \Downarrow R$.

For this example, we define the program relation $\preccurlyeq$ by

$$M \preccurlyeq M' \quad \text{if and only if} \quad \forall V : \mathbf{bool}.\ (M \Downarrow \mathbf{return}\, V) \ \Rightarrow \ (M' \Downarrow \mathbf{return}\, V)$$

so that $M \preccurlyeq^{\Gamma}_{\mathrm{ctx}} M'$ informally means if a program containing M terminates with some result then the same program with M' instead of M terminates with the same result.

Example 2.4 (Nondeterminism)**.** For our third example, we consider finite nondeterminism. Again call-by-value and call-by-name have different behaviour, but any result of a call-by-value execution is also a result of a call-by-name execution (if suitable nondeterministic choices are made).

We consider CBPV without recursion, but augmented with computations $\mathbf{fail}_{\underline{C}}$ for nullary nondeterministic choice and $M \mathbf{\ or\ } N$ for binary nondeterministic choice between computations; the typing and evaluation rules are standard:

$$\frac{}{\Gamma \vdash_c \mathbf{fail}_{\underline{C}} : \underline{C}} \qquad \frac{\Gamma \vdash_c M : \underline{C} \qquad \Gamma \vdash_c N : \underline{C}}{\Gamma \vdash_c M \mathbf{\ or\ } N : \underline{C}} \qquad \frac{M \Downarrow R}{M \mathbf{\ or\ } N \Downarrow R} \qquad \frac{N \Downarrow R}{M \mathbf{\ or\ } N \Downarrow R}$$

(There is no R such that $\mathbf{fail}_{\underline{C}} \Downarrow R$.) The computation $\mathbf{fail}_{\underline{C}}$ is the unit for **or**, so $\mathbf{fail}_{\underline{C}} \mathbf{\ or\ } M$ and $M \mathbf{\ or\ } \mathbf{fail}_{\underline{C}}$ have the same behaviour as M. For each closed computation $M : \mathbf{F}\, A$ there might be zero, one or several values $V : A$ such that $M \Downarrow \mathbf{return}\, V$.

We similarly include nullary and binary nondeterminism in the source language, and extend the call-by-value and call-by-name translations:

$$e \ ::= \ \ldots \mid \mathbf{fail}_\tau \mid e \mathbf{\ or\ } e' \qquad \frac{}{\Gamma \vdash \mathbf{fail}_\tau : \tau} \qquad \frac{\Gamma \vdash e : \tau \qquad \Gamma \vdash e' : \tau}{\Gamma \vdash e \mathbf{\ or\ } e' : \tau}$$

$$\begin{aligned} \mathcal{V}(\!|\mathbf{fail}_\tau|\!) &= \mathbf{fail}_{\mathbf{F}\, (\mathcal{V}(\!|\tau|\!))} & \mathcal{N}(\!|\mathbf{fail}_\tau|\!) &= \mathbf{fail}_{\mathcal{N}(\!|\tau|\!)} \\ \mathcal{V}(\!|e \mathbf{\ or\ } e'|\!) &= \mathcal{V}(\!|e|\!) \mathbf{\ or\ } \mathcal{V}(\!|e'|\!) & \mathcal{N}(\!|e \mathbf{\ or\ } e'|\!) &= \mathcal{N}(\!|e|\!) \mathbf{\ or\ } \mathcal{N}(\!|e'|\!) \end{aligned}$$

As an example, evaluating the expression $e = (\lambda x.\, \mathbf{if}\ x\ \mathbf{then}\ x\ \mathbf{else}\ \mathbf{true})(\mathbf{true} \mathbf{\ or\ } \mathbf{false})$ under call-by-value necessarily results in **true**, but under call-by-name we can also get **false**. (We have $\mathcal{V}(\!|e|\!) \not\Downarrow \mathbf{return}\, \mathbf{false}$ but $\mathcal{N}(\!|e|\!) \Downarrow \mathbf{return}\, \mathbf{false}$.)

For nondeterminism, we define $\preccurlyeq$ in the same way as our divergence example:

$$M \preccurlyeq M' \quad \text{if and only if} \quad \forall V : \mathbf{bool}.\ (M \Downarrow \mathbf{return}\, V) \ \Rightarrow \ (M' \Downarrow \mathbf{return}\, V)$$

This captures the property that any result that arises from an execution of M (which may involve call-by-value) might arise from an execution of M' (which may involve call-by-name).

Example 2.5 (Immutable state)**.** Finally, we consider the basic languages enriched with an extra construct for getting the value of a immutable state whose value is either *true* or *false*. Once again we do not expect there to be any difference between call-by-value and call-by-name, and it is indeed the case that if e is a closed expression of type **bool**, then call-by-value and call-by-name evaluations of e have the same behaviour (this is an instance of Corollary 4.8). Notably however, the model we use for this example fails to satisfy the assumptions of our main theorem (Theorem 6.2).

We augment CBPV with a computation **get**. This gets the value of the state, producing either **true** or **false**.

$$\frac{}{\Gamma \vdash_c \mathbf{get} : \mathbf{F}\,\mathbf{bool}}$$

Big-step evaluation has a slightly different form in this case. We write $M \Downarrow_b R$ to mean M evaluates to R when the state is $b \in \{\mathit{true}, \mathit{false}\}$. The rules are those of Figure 2 (with the subscript b added), plus

$$\frac{}{\mathbf{get} \Downarrow_{\mathit{true}} \mathbf{return}\,\mathbf{true}} \qquad \frac{}{\mathbf{get} \Downarrow_{\mathit{false}} \mathbf{return}\,\mathbf{false}}$$

Again we extend the source language, and also the call-by-value and call-by-name translations:

$$e ::= \ldots \mid \mathbf{get} \qquad \frac{}{\Gamma \vdash \mathbf{get} : \mathbf{bool}} \qquad \mathcal{V}(\!|\mathbf{get}|\!) = \mathcal{N}(\!|\mathbf{get}|\!) = \mathbf{get}$$

We define the program relation $\preccurlyeq$ as follows:

$$M \preccurlyeq M' \text{ if and only if } \forall b \in \{\mathit{true}, \mathit{false}\}.\, \exists V : \mathbf{bool}.\, (M \Downarrow_b \mathbf{return}\, V) \wedge (M' \Downarrow_b \mathbf{return}\, V)$$

3. Order-enriched denotational semantics

We give a denotational semantics for CBPV, which we use to prove instances of $\preccurlyeq_{\mathrm{ctx}}$. Since $\preccurlyeq_{\mathrm{ctx}}$ is not in general symmetric, we use *order-enriched* models, which come with partial orders $\sqsubseteq$ between denotations. In an *adequate* model, $[\![M]\!] \sqsubseteq [\![N]\!]$ implies $M \preccurlyeq_{\mathrm{ctx}} N$. Our semantics is based on Levy's *algebra models* [Lev06] for CBPV, in which each computation type is interpreted as a monad algebra. (We restrict to algebra models for simplicity. Other forms of model, such as *adjunction models* [Lev03] can be used for the same purpose.)

3.1. Order-enriched categories and strong monads.

We define the basic categorical notions we need for the rest of the paper. We assume no knowledge of enriched category theory; instead we give the relevant order-enriched (specifically **Poset**-enriched) definitions here. (We do however assume some basic ordinary category theory.)

Definition 3.1. A **Poset**-*category* $\mathbf{C}$ is an ordinary category, together with a partial order $\sqsubseteq$ on each hom-set $\mathbf{C}(X, Y)$, such that composition is monotone.

If $\mathbf{C}$ is a **Poset**-category, we refer to the ordinary category as the *underlying* ordinary category, and write $|\mathbf{C}|$ for the class of objects.

Example 3.2. We use the following three **Poset**-categories.

Poset-category $\mathbf{C}$	Objects $X \in \lvert\mathbf{C}\rvert$	Morphisms $f : X \to Y$	Order $f \sqsubseteq f'$
Set	sets	functions	equality
Poset	posets	monotone functions	pointwise
ω**Cpo**	ωcpos	ω-continuous functions	pointwise

In each case, composition and identities are defined in the usual way. For **Set**, since the hom-posets $\mathbf{Set}(X, Y)$ are discrete, all of the **Poset**-enriched definitions coincide with the ordinary (unenriched) definitions. The objects of $\omega\mathbf{Cpo}$ are posets $(X, \sqsubseteq)$ for which $\sqsubseteq$ is ω-*complete*, i.e. for which every ω-chain $x_0 \sqsubseteq x_1 \sqsubseteq \cdots$ has a least upper bound $\bigsqcup x$. Morphisms are ω-*continuous* functions, i.e. monotone functions that preserve least upper bounds of ω-chains.

Let $\mathbf{C}$ be a **Poset**-category. We say that $\mathbf{C}$ is *cartesian* when its underlying category has a terminal object 1 and binary products $X_1 \times X_2$, such that the pairing functions $\langle -, - \rangle : \mathbf{C}(W, X_1) \times \mathbf{C}(W, X_2) \to \mathbf{C}(W, X_1 \times X_2)$ are monotone. We write $\pi_i : X_1 \times X_2 \to X_i$ for the projections from a product, and write $\langle\rangle_X : X \to 1$ for the unique map into the terminal object. In every cartesian category, there are canonical *associativity* isomorphisms $assoc_{X_1,X_2,X_3} : (X_1 \times X_2) \times X_3 \to X_1 \times (X_2 \times X_3)$. We say that $\mathbf{C}$ is *cartesian closed* when it is cartesian and its underlying category has exponentials $X \Rightarrow Y$ for which the currying functions $\Lambda : \mathbf{C}(W \times X, Y) \to \mathbf{C}(W, X \Rightarrow Y)$ are monotone. (It follows that the uncurrying functions $\Lambda^{-1} : \mathbf{C}(W, X \Rightarrow Y) \to \mathbf{C}(W \times X, Y)$ are also monotone.) We write $ev_{X,Y}$ for the *evaluation* morphism $\Lambda^{-1} id : (X \Rightarrow Y) \times X \to Y$. Binary *coproducts* in $\mathbf{C}$ are just binary coproducts in the underlying ordinary category, except that the copairing functions $[-, -] : \mathbf{C}(X_1, W) \times \mathbf{C}(X_2, W) \to \mathbf{C}(X_1 + X_2, W)$ are required to be monotone. We write $inl : X_1 \to X_1 + X_2$ and $inr : X_2 \to X_1 + X_2$ for the coprojections. The **Poset**-categories **Set**, **Poset**, and $\omega\mathbf{Cpo}$ are all cartesian closed, and have binary coproducts given by disjoint union.

We interpret computation types as (Eilenberg–Moore) algebras for an order-enriched monad T, which we need to be *strong* (just as models of Moggi's monadic metalanguage [Mog91] use a strong monad). The definitions of strong **Poset**-monad and of T-algebra we give are slightly non-standard, but are equivalent to the standard ones (see for example [MU22]). In particular, it is more convenient for us to bake the strength into the (Kleisli) extension of the monad instead of having a separate strength.

Definition 3.3 (Strong **Poset**-monad)**.** Let $\mathbf{C}$ be a cartesian **Poset**-category. A *strong* **Poset**-*monad* T on $\mathbf{C}$ consists of:

- an object $TX \in |\mathbf{C}|$ for each $X \in |\mathbf{C}|$;
- a morphism $\eta_X : X \to TX$ for each $X \in |\mathbf{C}|$ (the *unit*);
- a monotone function $(-)^{\dagger^{W \times \square}} : \mathbf{C}(W \times X, TY) \to \mathbf{C}(W \times TX, TY)$ (*Kleisli extension*) for each $W, X, Y \in |\mathbf{C}|$.

These are required to satisfy the following four laws.[2]

- Naturality of extension in W:
$$f^{\dagger^{W \times \square}} \circ (w \times id_{TX}) \;=\; (f \circ (w \times id_X))^{\dagger^{W' \times \square}}$$
for all $f : W \times X \to TY$ and $w : W' \to W$.
- Left unit:
$$f^{\dagger^{W \times \square}} \circ (id_W \times \eta_X) \;=\; f$$
for all $f : W \times X \to TY$.

[2]The conference version [MM22] of this paper incorrectly omits naturality in W from the definition of strong **Poset**-monad and from the definition of Eilenberg–Moore algebra. (Naturality in W is required in [MU22, Definition 4.1].)

- Right unit:
$$(\eta_X \circ \pi_2)^{\dagger^{1\times\Box}} = \pi_2$$
for all $X \in |\mathbf{C}|$.
- Associativity:
$$(g^{\dagger^{W'\times\Box}} \circ (id_{W'} \times f) \circ assoc)^{\dagger^{(W'\times W)\times\Box}} = g^{\dagger^{W'\times\Box}} \circ (id_{W'} \times f^{\dagger^{W\times\Box}}) \circ assoc$$
for all $f : W \times X \to TY$ and $g : W' \times Y \to TZ$.

Specializing the Kleisli extension of T to $W = 1$ produces a (non-strong) extension operator $(-)^\dagger : \mathbf{C}(X, TY) \to \mathbf{C}(TX, TY)$, satisfying the usual monad laws:
$$f^\dagger \circ \eta_X = f \qquad {\eta_X}^\dagger = id_X \qquad (g^\dagger \circ f)^\dagger = g^\dagger \circ f^\dagger$$
We use this to define, for every $f : X \to Y$, a morphism $Tf : TX \to TY$ by $Tf = (\eta_Y \circ f)^\dagger$. The latter definition makes T into a **Poset**-*functor*: the mapping $f \mapsto Tf$ is monotone, and preserves identities and composition. The definition of T on morphisms also ensures that unit and Kleisli extension of T satisfy the following naturality laws:
$$Tf \circ \eta_X = \eta_Y \circ f \quad (Tg \circ f)^{\dagger^{W\times\Box}} = Tg \circ f^{\dagger^{W\times\Box}} \quad (f \circ (id_W \times g))^{\dagger^{W\times\Box}} = f^{\dagger^{W\times\Box}} \circ (id_W \times Tg)$$
In the notation $f^{\dagger^{W\times\Box}} : W \times TX \to TY$, the square $\Box$ is intended to indicate the position of T in the domain. Since products are symmetric, choosing to put T to the right of W is arbitrary. We construct a Kleisli extension operator with the square to the left as follows:
$$f^{\dagger^{\Box\times W}} = (f \circ \langle \pi_2, \pi_1 \rangle)^{\dagger^{W\times\Box}} \circ \langle \pi_2, \pi_1 \rangle : TX \times W \to TY \qquad (\text{where } f : X \times W \to TY)$$
We also define two natural transformations for sequencing of computations: seq^{L} for left-to-right and seq^{R} for right-to-left, as follows.
$$\begin{aligned} seq^{\mathrm{L}}_{X_1,X_2} &= ({\eta_{X_1\times X_2}}^{\dagger^{X_1\times\Box}})^{\dagger^{\Box\times TX_2}} &&: TX_1 \times TX_2 \to T(X_1 \times X_2) \\ seq^{\mathrm{R}}_{X_1,X_2} &= ({\eta_{X_1\times X_2}}^{\dagger^{\Box\times X_2}})^{\dagger^{TX_1\times\Box}} &&: TX_1 \times TX_2 \to T(X_1 \times X_2) \end{aligned}$$
We further define an *effectful* pairing operation $\langle\!\langle -, - \rangle\!\rangle$:
$$\langle\!\langle f_1, f_2 \rangle\!\rangle = seq^{\mathrm{L}}_{X_1,X_2} \circ \langle f_1, f_2 \rangle \;:\; W \to T(X_1 \times X_2) \qquad (\text{where } f_i : W \to TX_i)$$
This evaluates from left to right; we do not need the right-to-left version.

Definition 3.4 (Eilenberg–Moore algebra)**.** Let T be a strong **Poset**-monad on a cartesian **Poset**-category $\mathbf{C}$. A T-*algebra* $\mathsf{Z} = (Z, (-)^\ddagger)$ is a pair of:
- an object $Z \in |\mathbf{C}|$ (the *carrier*);
- a monotone function $(-)^{\ddagger^{W\times\Box}} : \mathbf{C}(W \times X, Z) \to \mathbf{C}(W \times TX, Z)$ (the *extension* operator) for each $W, X \in |\mathbf{C}|$.

These are required to satisfy the following three laws.
- Naturality in W:
$$f^{\ddagger^{W\times\Box}} \circ (w \times id_{TX}) \;=\; (f \circ (w \times id_X))^{\ddagger^{W'\times\Box}}$$
for all $f : W \times X \to Z$ and $w : W' \to W$.

- Left unit:
$$f^{\ddagger^{W\times\Box}} \circ (id_W \times \eta_X) \;=\; f$$
for all $f : W \times X \to Z$.
- Associativity:
$$(g^{\ddagger^{W'\times\Box}} \circ (id_{W'} \times f) \circ assoc)^{\ddagger^{(W\times W')\times\Box}} = g^{\ddagger^{W'\times\Box}} \circ (id_{W'} \times f^{\dagger^{W\times\Box}}) \circ assoc$$
for all $f : W \times X \to TY$ and $g : W' \times Y \to Z$.

For each T-algebra Z, we write $U_{\mathsf{T}}\mathsf{Z}$ for the carrier $Z \in |\mathbf{C}|$.

Just as for the extension operator of a strong **Poset**-monad, we specialize the extension operator of a T-algebra to $W = 1$ and obtain a (non-strong) extension operator $(-)^{\ddagger} : \mathbf{C}(X, Z) \to \mathbf{C}(TX, Z)$. We also have an extension operator with reversed products, written $(-)^{\ddagger^{\Box\times W}} : \mathbf{C}(X \times W, Z) \to \mathbf{C}(TX \times W, Z)$.

The following constructions of algebras are standard.

Definition 3.5. Let T be a strong **Poset**-monad on a cartesian closed **Poset**-category $\mathbf{C}$.
- The *free* T-algebra on an object $X \in |\mathbf{C}|$ has carrier TX and extension operator $(-)^{\dagger}$.
- If Z_1 and Z_2 are T-algebras, then their *product* $\mathsf{Z}_1 \times \mathsf{Z}_2$ is the T-algebra with carrier $Z_1 \times Z_2$, and extension operator
$$f^{\ddagger^{W\times\Box}} = \langle (\pi_1 \circ f)^{\ddagger^{W\times\Box}}, (\pi_2 \circ f)^{\ddagger^{W\times\Box}} \rangle$$
- If $Y \in |\mathbf{C}|$ and Z is a T-algebra, then their *power* $Y \Rightarrow \mathsf{Z}$ is the T-algebra with carrier $Y \Rightarrow Z$ and extension operator
$$f^{\ddagger^{W\times\Box}} = \Lambda((\Lambda^{-1} f \circ \beta_{W,Y,X})^{\ddagger^{(W\times Y)\times\Box}} \circ \beta_{W,TX,Y})$$
where $\beta_{X_1,X_2,X_3} = \langle \langle \pi_1 \circ \pi_1, \pi_2 \rangle, \pi_2 \circ \pi_1 \rangle : (X_1 \times X_2) \times X_3 \to (X_1 \times X_3) \times X_2$.

We use these constructions to interpret CBPV computation types: returner types $\mathbf{F}\,A$ are interpreted as free T-algebras, product types $\underline{C}_1 \mathbin{\&} \underline{C}_2$ are interpreted as product T-algebras, and function types $A \to \underline{C}$ are interpreted as power T-algebras.

3.2. **Models of CBPV.** We define the notion of (order-enriched, algebra) *model* as follows.

Definition 3.6. A *model* $\mathcal{M} = (\mathbf{C}, \mathsf{T})$ of CBPV consists of
- a cartesian closed **Poset**-category $\mathbf{C}$ that admits the coproduct $2 = 1 + 1$;
- a strong **Poset**-monad T on $\mathbf{C}$.

Given a model $\mathcal{M} = (\mathbf{C}, \mathsf{T})$, the interpretation $[\![-]\!]$ of CBPV is defined in Figure 4. Value types A are interpreted as objects $[\![A]\!] \in |\mathbf{C}|$, while computation types $\underline{C}$ are interpreted as T-algebras. Typing contexts Γ are interpreted as objects $[\![\Gamma]\!] \in \mathbf{C}$ using the cartesian structure of $\mathbf{C}$; if $(x : A) \in \Gamma$ then we write π_x for the corresponding projection $[\![\Gamma]\!] \to [\![A]\!]$. Values $\Gamma \vdash V : A$ (respectively computations $\Gamma \vdash_c M : \underline{C}$) are interpreted as morphisms $[\![\Gamma \vdash V : A]\!]$ (resp. $[\![\Gamma \vdash_c M : \underline{C}]\!]$) in $\mathbf{C}$; we often omit the typing context and type when writing these. Programs $\diamond \vdash_c M : \mathbf{bool}$ are therefore interpreted as morphisms $[\![M]\!] : 1 \to T2$. To interpret **if**, we use the fact that, since $\mathbf{C}$ is cartesian closed, products distribute over the coproduct $2 = 1 + 1$. This means that for every $W \in |\mathbf{C}|$, the coproduct $W + W$ also exists in $\mathbf{C}$, and the canonical morphism
$$W + W \xrightarrow{[\langle id_W, inl \circ \langle\rangle_W \rangle, \langle id_W, inr \circ \langle\rangle_W \rangle]} W \times 2$$

$\boxed{\mathbf{C}\text{-object } [\![A]\!]}$

$$
\begin{aligned}
[\![\mathbf{unit}]\!] &= 1 \\
[\![A_1 \times A_2]\!] &= [\![A_1]\!] \times [\![A_2]\!] \\
[\![\mathbf{bool}]\!] &= 2 \quad (= 1{+}1) \\
[\![\mathbf{U}\,\underline{C}]\!] &= U_{\mathsf{T}}[\![\underline{C}]\!]
\end{aligned}
$$

$\boxed{\mathsf{T}\text{-algebra } [\![\underline{C}]\!]}$

$$
\begin{aligned}
[\![\underline{C}_1 \precsim \underline{C}_2]\!] &= [\![\underline{C}_1]\!] \times [\![\underline{C}_2]\!] \\
[\![A \to \underline{C}]\!] &= [\![A]\!] \Rightarrow [\![\underline{C}]\!] \\
[\![\mathbf{F}\,A]\!] &= F_{\mathsf{T}}[\![A]\!]
\end{aligned}
$$

$\boxed{\mathbf{C}\text{-object } [\![\Gamma]\!]}$

$$
\begin{aligned}
[\![\diamond]\!] &= 1 \\
[\![\Gamma, x : A]\!] &= [\![\Gamma]\!] \times [\![A]\!]
\end{aligned}
$$

$\boxed{[\![\Gamma \vdash V : A]\!] : [\![\Gamma]\!] \to [\![A]\!]}$

$$
\begin{aligned}
[\![x]\!] &= \pi_x \\
[\![()]\!] &= \langle\rangle_{[\![\Gamma]\!]} \\
[\![(V_1, V_2)]\!] &= \langle [\![V_1]\!], [\![V_2]\!] \rangle \\
[\![\mathbf{true}]\!] &= \mathit{inl} \circ \langle\rangle_{[\![\Gamma]\!]} \\
[\![\mathbf{false}]\!] &= \mathit{inr} \circ \langle\rangle_{[\![\Gamma]\!]} \\
[\![\mathbf{thunk}\,M]\!] &= [\![M]\!]
\end{aligned}
$$

$\boxed{[\![\Gamma \vdash_c M : \underline{C}]\!] : [\![\Gamma]\!] \to U_{\mathsf{T}}[\![\underline{C}]\!]}$

$$
\begin{aligned}
[\![\lambda\{1.\, M_1, 2.\, M_2\}]\!] &= \langle [\![M_1]\!], [\![M_2]\!] \rangle \\
[\![i\text{`}M]\!] &= \pi_i \circ [\![M]\!] \qquad (i \in \{1,2\}) \\
[\![\lambda x\!:\!A.\, M]\!] &= \Lambda[\![M]\!] \\
[\![V\text{`}M]\!] &= \Lambda^{-1}[\![M]\!] \circ \langle \mathit{id}, [\![V]\!] \rangle \\
[\![\mathbf{return}\,V]\!] &= \eta \circ [\![V]\!] \\
[\![M \ \mathbf{to}\ x.\, N]\!] &= [\![N]\!]^{\ddagger^{[\![\Gamma]\!] \times \square}} \circ \langle \mathit{id}, [\![M]\!] \rangle \\
[\![\mathbf{match}\,V\ \mathbf{with}\ (x,y).\, M]\!] &= [\![M]\!] \circ \mathit{assoc}^{-1} \circ \langle \mathit{id}, [\![V]\!] \rangle \\
[\![\mathbf{if}\ V\ \mathbf{then}\ M_1\ \mathbf{else}\ M_2]\!] &= [[\![M_1]\!], [\![M_2]\!]] \circ \mathit{dist} \circ \langle \mathit{id}, [\![V]\!] \rangle \\
[\![\mathbf{force}\,V]\!] &= [\![V]\!]
\end{aligned}
$$

FIGURE 4. Denotational semantics of CBPV

has an inverse $\mathit{dist}_W : W \times 2 \to W + W$.

By composing the semantics of CBPV with the two translations of the source language, we obtain a call-by-value semantics $\mathcal{V}[\![-]\!] = [\![\mathcal{V}(\!|-|\!)]\!]$ and a call-by-name semantics $\mathcal{N}[\![-]\!] = [\![\mathcal{N}(\!|-|\!)]\!]$ of the source language. For convenience, we spell out these composed semantics in Figure 5.

We use the denotational semantics as a tool for proving instances of contextual preorders; for this we need *adequacy*.

Definition 3.7. A model of CBPV is *adequate* with respect to a given program relation $\preccurlyeq$ if for all computations $\Gamma \vdash_c M : \underline{C}$ and $\Gamma \vdash_c M' : \underline{C}$ we have

$$[\![\Gamma \vdash_c M : \underline{C}]\!] \sqsubseteq [\![\Gamma \vdash_c M' : \underline{C}]\!] \quad \Rightarrow \quad M \preccurlyeq^{\Gamma}_{\mathrm{ctx}} M'$$

3.3. **Examples.** We give four different models, one for each of the four examples in Section 2.2. Each model is adequate with respect to the corresponding definition of $\preccurlyeq$; the proof in each case is a standard *logical relations* argument (e.g. [Win93]).

Example 3.8. For CBPV with no side-effects, we use $\mathbf{C} = \mathbf{Set}$. The strong **Poset**-monad T is the identity on **Set**. Each T-algebra Z is completely determined by its carrier Z; the

$$\boxed{\text{type } \tau \;\mapsto\; \text{object } \mathcal{V}[\![\tau]\!] \in |\mathbf{C}|}$$

$$\begin{array}{rcl} \mathbf{unit} & \mapsto & 1 \\ \tau_1 \times \tau_2 & \mapsto & \mathcal{V}[\![\tau_1]\!] \times \mathcal{V}[\![\tau_2]\!] \\ \mathbf{bool} & \mapsto & 2 \\ \tau \to \tau' & \mapsto & \mathcal{V}[\![\tau]\!] \Rightarrow T(\mathcal{V}[\![\tau']\!]) \end{array}$$

$$\boxed{\text{typing context } \Gamma \;\mapsto\; \text{object } \mathcal{V}[\![\Gamma]\!] \in |\mathbf{C}|}$$

$$\begin{array}{rcl} \diamond & \mapsto & 1 \\ \Gamma, x : \tau & \mapsto & \mathcal{V}[\![\Gamma]\!] \times \mathcal{V}[\![\tau]\!] \end{array}$$

$$\boxed{\text{expression } \Gamma \vdash e : \tau \;\mapsto\; \text{morphism } \mathcal{V}[\![e]\!] : \mathcal{V}[\![\Gamma]\!] \to T(\mathcal{V}[\![\tau]\!]) \text{ in } \mathbf{C}}$$

$$\begin{array}{rcl} x & \mapsto & \pi_x \\ () & \mapsto & \eta_1 \circ \langle\rangle_{\mathcal{V}[\![\Gamma]\!]} \\ (e_1, e_2) & \mapsto & \langle\!\langle \mathcal{V}[\![e_1]\!], \mathcal{V}[\![e_2]\!] \rangle\!\rangle \\ \mathbf{fst}\, e & \mapsto & T\pi_1 \circ \mathcal{V}[\![e]\!] \\ \mathbf{snd}\, e & \mapsto & T\pi_2 \circ \mathcal{V}[\![e]\!] \\ \mathbf{true} & \mapsto & \eta_2 \circ \mathit{inl} \circ \langle\rangle_{\mathcal{V}[\![\Gamma]\!]} \\ \mathbf{false} & \mapsto & \eta_2 \circ \mathit{inr} \circ \langle\rangle_{\mathcal{V}[\![\Gamma]\!]} \\ \mathbf{if}\ e_0\ \mathbf{then}\ e_1\ \mathbf{else}\ e_2 & \mapsto & ([\mathcal{V}[\![e_1]\!], \mathcal{V}[\![e_2]\!]] \circ \mathit{dist})^{\dagger^{\mathcal{V}[\![\Gamma]\!] \times \square}} \circ \langle \mathit{id}_{\mathcal{V}[\![\Gamma]\!]}, \mathcal{V}[\![e_0]\!] \rangle \\ \lambda x\!:\!\tau.\, e & \mapsto & \eta_{\mathcal{V}[\![\tau \to \tau']\!]} \circ \Lambda(\mathcal{V}[\![e]\!]) \\ e\, e' & \mapsto & \mathit{ev}^\dagger \circ \langle\!\langle \mathcal{V}[\![e]\!], \mathcal{V}[\![e']\!] \rangle\!\rangle \end{array}$$

(A) Call-by-value semantics $\mathcal{V}[\![-]\!]$

$$\boxed{\text{type } \tau \;\mapsto\; \mathsf{T}\text{-algebra } \mathcal{N}[\![\tau]\!]}$$

$$\begin{array}{rcl} \mathbf{unit} & \mapsto & F_{\mathsf{T}} 1 \\ \tau_1 \times \tau_2 & \mapsto & \mathcal{N}[\![\tau_1]\!] \times \mathcal{N}[\![\tau_2]\!] \\ \mathbf{bool} & \mapsto & F_{\mathsf{T}} 2 \\ \tau \to \tau' & \mapsto & (U_{\mathsf{T}}(\mathcal{N}[\![\tau]\!])) \Rightarrow \mathcal{N}[\![\tau']\!] \end{array}$$

$$\boxed{\text{typing context } \Gamma \;\mapsto\; \text{object } \mathcal{N}[\![\Gamma]\!] \in |\mathbf{C}|}$$

$$\begin{array}{rcl} \diamond & \mapsto & 1 \\ \Gamma, x : \tau & \mapsto & \mathcal{N}[\![\Gamma]\!] \times U_{\mathsf{T}}(\mathcal{N}[\![\tau]\!]) \end{array}$$

$$\boxed{\text{expression } \Gamma \vdash e : \tau \;\mapsto\; \text{morphism } \mathcal{N}[\![e]\!] : \mathcal{N}[\![\Gamma]\!] \to U_{\mathsf{T}}(\mathcal{N}[\![\tau]\!]) \text{ in } \mathbf{C}}$$

$$\begin{array}{rcl} x & \mapsto & \pi_x \\ () & \mapsto & \eta_1 \circ \langle\rangle_{\mathcal{N}[\![\Gamma]\!]} \\ (e_1, e_2) & \mapsto & \langle \mathcal{N}[\![e_1]\!], \mathcal{N}[\![e_2]\!] \rangle \\ \mathbf{fst}\, e & \mapsto & \pi_1 \circ \mathcal{N}[\![e]\!] \\ \mathbf{snd}\, e & \mapsto & \pi_2 \circ \mathcal{N}[\![e]\!] \\ \mathbf{true} & \mapsto & \eta_2 \circ \mathit{inl} \circ \langle\rangle_{\mathcal{N}[\![\Gamma]\!]} \\ \mathbf{false} & \mapsto & \eta_2 \circ \mathit{inr} \circ \langle\rangle_{\mathcal{N}[\![\Gamma]\!]} \\ \mathbf{if}\ e_0\ \mathbf{then}\ e_1\ \mathbf{else}\ e_2 & \mapsto & ([\mathcal{N}[\![e_1]\!], \mathcal{N}[\![e_2]\!]] \circ \mathit{dist})^{\ddagger^{\mathcal{N}[\![\Gamma]\!] \times \square}} \circ \langle \mathit{id}_{\mathcal{N}[\![\Gamma]\!]}, \mathcal{N}[\![e_0]\!] \rangle \\ \lambda x\!:\!\tau.\, e & \mapsto & \Lambda(\mathcal{N}[\![e]\!]) \\ e\, e' & \mapsto & \mathit{ev} \circ \langle \mathcal{N}[\![e]\!], \mathcal{N}[\![e']\!] \rangle \end{array}$$

(B) Call-by-name semantics $\mathcal{N}[\![-]\!]$

FIGURE 5. Denotational semantics of call-by-value and call-by-name

extension operator $(-)^{\ddagger^{W\times\Box}} : \mathbf{Set}(W \times X, Z) \to \mathbf{Set}(W \times X, Z)$ is necessarily the identity. The interpretation $[\![M]\!]$ of each program M is just an element of 2.

Example 3.9. For divergence, we use $\mathbf{C} = \omega\mathbf{Cpo}$. The strong **Poset**-monad T freely adjoins a least element $\bot$ to each $\omega\mathbf{Cpo}$. The unit η_X is the inclusion $X \hookrightarrow TX$, while Kleisli extension is given by

$$f^{\dagger^{W\times\Box}}(w,x) = \begin{cases} \bot & \text{if } x = \bot \\ f(w,x) & \text{otherwise} \end{cases}$$

A T-algebra Z is equivalently an $\omega\mathbf{Cpo}$ Z with a least element $\bot \in Z$. The extension operator is completely determined once the carrier is fixed; it is analogous to $(-)^\dagger$.

In this case, the product $Z_1 \times Z_2$ is the set of pairs ordered componentwise, and the exponential $Y \Rightarrow Z$ is the set of set of ω-continuous functions ordered pointwise. Hence $Z_1 \times Z_2$ has a least element $(\bot, \bot)$ (so forms a T-algebra) whenever Z_1 and Z_2 have least elements, and $Y \Rightarrow Z$ has a least element (the constantly-$\bot$ function) whenever Z has a least element.

If Z is a T-algebra, then every ω-continuous function $f : Z \to Z$ has a least fixed point $\mathit{fix}\, f = \bigsqcup_{n\in\mathbb{N}} f^n \bot \in Z$. These enable us to interpret recursive computations, by defining $[\![\mathbf{rec}\, x : \mathbf{U}\,\underline{C}.\, M]\!]\rho = \mathit{fix}\big(\lambda x.\, [\![M]\!](\rho, x)\big)$. The interpretation $[\![M]\!]$ of a program $M : \mathbf{F}\,\mathbf{bool}$ is either $\bot$ (signifying divergence), or one of the two elements of 2.

Example 3.10. For finite nondeterminism, we use $\mathbf{C} = \mathbf{Poset}$. The strong **Poset**-monad T freely adds finite joins to each poset. It is defined by

$$TX = (\{{\downarrow}S' \mid S' \in \mathcal{P}_{\mathrm{fin}}X\}, \subseteq) \qquad \eta_X\, x = {\downarrow}\{x\} \qquad f^{\dagger^{W\times\Box}}(w,S) = \textstyle\bigcup_{x\in S} f(w,x)$$

where $\mathcal{P}_{\mathrm{fin}}X$ is the set of finite subsets of X, and ${\downarrow}S' = \{x \in X \mid \exists x' \in S'.\, x \sqsubseteq x'\}$ is the *downwards-closure* of $S' \subseteq X$. Each T-algebra is again completely determined by its carrier; a T-algebra Z is equivalently a poset Z that has finite joins. The extension operator is necessarily given by $f^{\ddagger^{W\times\Box}}(w,S) = \bigsqcup_{x\in S} f(w,x)$. (The latter join exists because S is the downwards-closure of a finite set, even though S itself might not be finite.) The product $Z_1 \times Z_2$ is the set of pairs ordered componentwise, with joins given by $\bigsqcup_i (z_i, z_i') = (\bigsqcup_i z_i, \bigsqcup_i z_i')$. The power $Y \Rightarrow Z$ is the set of monotone functions ordered pointwise, with joins given by $(\bigsqcup_i f_i)x = \bigsqcup_i (f_i x)$.

We interpret nondeterministic computations using nullary and binary joins:

$$[\![\mathbf{fail}_{\underline{C}}]\!]\rho = \bot \qquad [\![M \,\mathbf{or}\, N]\!]\rho = [\![M]\!]\rho \sqcup [\![N]\!]\rho$$

The interpretation $[\![M]\!]$ of a program $M : \mathbf{F}\,\mathbf{bool}$ is one of the four subsets of 2.

Example 3.11. For immutable state, we use $\mathbf{C} = \mathbf{Set}$, with the *reader* monad

$$TX = (2 \Rightarrow X) \quad \eta_X x = \lambda b.\, x \quad f^{\dagger^{W\times\Box}}(w,t) = \lambda b.\, f(w, t\, b)\, b$$

where $2 = \{\mathit{true}, \mathit{false}\}$. The CBPV computation **get** is interpreted as

$$[\![\mathbf{get}]\!]\rho = \lambda b.\, b$$

$$\boxed{f^v \ \mathcal{R}[\![\tau]\!]_W \ f^n \text{ (where } f^v : W \to T(\mathcal{V}[\![\tau]\!]) \text{ and } f^n : W \to U_{\mathsf{T}}(\mathcal{N}[\![\tau]\!]))}$$

$$\begin{aligned}
f^v \ \mathcal{R}[\![\mathbf{unit}]\!]_W \ f^n &\text{ iff } f^v \sqsubseteq f^n \\
f^v \ \mathcal{R}[\![\tau_1 \times \tau_2]\!]_W \ f^n &\text{ iff } (T\pi_1 \circ f^v) \ \mathcal{R}[\![\tau_1]\!]_W \ (\pi_1 \circ f^n) \ \wedge \ (T\pi_2 \circ f^v) \ \mathcal{R}[\![\tau_2]\!]_W \ (\pi_2 \circ f^n) \\
f^v \ \mathcal{R}[\![\mathbf{bool}]\!]_W \ f^n &\text{ iff } f^v \sqsubseteq f^n \\
f^v \ \mathcal{R}[\![\tau \to \tau']\!]_W \ f^n &\text{ iff } \forall W', w : W' \to W, g^v, g^n. \\
&\qquad g^v \ \mathcal{R}[\![\tau]\!]_{W'} \ g^n \ \Rightarrow \ (ev^\dagger \circ \langle\!\langle f^v \circ w, g^v \rangle\!\rangle) \ \mathcal{R}[\![\tau']\!]_{W'} \ (ev \circ \langle f^n \circ w, g^n \rangle)
\end{aligned}$$

FIGURE 6. The relation between call-by-value and call-by-name

4. The relation between call-by-value and call-by-name

We now return to the main contribution of this paper: relating call-by-value with call-by-name. Recall the first step outlined in the introduction. We define a family of relations $\ltimes$ (Definition 4.1) that compare the observable behaviour of a computation of call-by-value type with a computation of call-by-name type. The main result of this section is that, under certain conditions on side-effects, we have

$$\mathcal{V}(\!|e|\!) \ltimes^\Gamma \mathcal{N}(\!|e|\!)$$

for all $\Gamma \vdash e : \tau$ (Theorem 4.7).

Since we work with the denotational semantics, instead of with the syntax directly, we define $\ltimes$ with respect to a fixed CBPV model $\mathcal{M}$ (and we write $\ltimes^\Gamma_{\mathcal{M}}$ to make $\mathcal{M}$ explicit). At the semantic level, we define a family of relations

$$f^v \ \mathcal{R}[\![\tau]\!]_W \ f^n$$

that relate elements f^v of $T(\mathcal{V}[\![\tau]\!])$ with elements f^n of $U_{\mathsf{T}}(\mathcal{N}[\![\tau]\!])$. Here by element we mean *generalized* element, so f^v and f^n are morphisms

$$f^v : W \to T(\mathcal{V}[\![\tau]\!]) \qquad f^n : W \to U_{\mathsf{T}}(\mathcal{N}[\![\tau]\!])$$

from some W.

We define $\mathcal{R}[\![\tau]\!]_W$ in the style of a logical relation, by induction on the type τ. The cases are listed in Figure 6. Informally, we have the following.

- For $\tau = \mathbf{unit}$ and $\tau = \mathbf{bool}$, we compare f^v and f^n directly using the order relation $\sqsubseteq$ on morphisms in $\mathbf{C}$. (We can do this because $T(\mathcal{V}[\![\tau]\!]) = U_{\mathsf{T}}(\mathcal{N}[\![\tau]\!])$.)
- For a product type $\tau_1 \times \tau_2$, we compare the first components and compare the second components. We get these components by composing with the call-by-value and call-by-name interpretations of the projections **fst** and **snd**.
- For a function type $\tau \to \tau'$, we relate f^v to f^n when these give related results when applied to related arguments. Here we use the call-by-value and call-by-name interpretations of application.

Note that in the function case we quantify over morphisms $w : W' \to W$ to permit varying *arities* W (cf. the Kripke logical relations of varying arity of [JT93]). (Precisely, this ensures that $\mathcal{R}[\![\tau]\!]$ is closed under precomposition with morphisms $w : W' \to W$, as in Lemma 4.5(1) below.)

We define $M^v \ltimes^\Gamma_{\mathcal{M}} M^n$, where $\mathcal{V}(\!|\Gamma|\!) \vdash_c M^v : \mathbf{F}\,(\mathcal{V}(\!|\tau|\!))$ and $\mathcal{N}(\!|\Gamma|\!) \vdash_c M^n : \mathcal{N}(\!|\tau|\!)$ are CBPV computations, in terms of $\mathcal{R}[\![-]\!]$. To state the definition, we need some more notation. Let

$\Gamma' = x_1 : A_1, \ldots, x_k : A_k$ be a CBPV context. Given a morphism $f_i : W \to [\![A_i]\!]$ for each $i \leq k$, we obtain a morphism $\langle f_i \rangle_i : W \to [\![\Gamma]\!]$, by iterated pairing. Given instead a morphism $f_i : W \to T[\![A_i]\!]$ for each i, we obtain a morphism $\langle\!\langle f_i \rangle\!\rangle_i : W \to T[\![\Gamma]\!]$ by iterated pairing and left-to-right evaluation:

$$\langle f_i \rangle_{i \leq k} = \begin{cases} \langle\rangle_W & \text{if } k = 0 \\ \langle \langle f_i \rangle_{i \leq (k-1)}, f_k \rangle & \text{if } k > 0 \end{cases} \qquad \langle\!\langle f_i \rangle\!\rangle_{i \leq k} = \begin{cases} \eta_1 \circ \langle\rangle_W & \text{if } k = 0 \\ \langle\!\langle \langle\!\langle f_i \rangle\!\rangle_{i \leq (k-1)}, f_k \rangle\!\rangle & \text{if } k > 0 \end{cases}$$

Definition 4.1. Let $\mathcal{M}$ be a CBPV model, and let

$$\mathcal{V}(\!|\Gamma|\!) \vdash_c M^v : \mathbf{F}\,(\mathcal{V}(\!|\tau'|\!)) \qquad \mathcal{N}(\!|\Gamma|\!) \vdash_c M^n : \mathcal{N}(\!|\tau'|\!)$$

be CBPV computations, where $\Gamma = x_1 : \tau_1, \ldots, x_k : \tau_k$. We write $M^v \ltimes^{\Gamma}_{\mathcal{M}} M^n$ when, for all objects $W \in |\mathbf{C}|$ and families of morphisms

$$(f^v_i : W \to T(\mathcal{V}[\![\tau_i]\!]))_i \qquad (f^n_i : W \to U_{\mathsf{T}}(\mathcal{N}[\![\tau_i]\!]))_i$$

we have

$$f^v_1 \; \mathcal{R}[\![\tau_1]\!]_W \; f^n_1 \;\wedge\; \cdots \;\wedge\; f^v_k \; \mathcal{R}[\![\tau_k]\!]_W \; f^n_k \quad \Rightarrow \quad ([\![M^v]\!]^\dagger \circ \langle\!\langle f^v_i \rangle\!\rangle_i) \; \mathcal{R}[\![\tau']\!]_W \; ([\![M^n]\!] \circ \langle f^n_i \rangle_i)$$

When $M^v : \mathbf{F}\,(\mathcal{V}(\!|\tau|\!))$ and $M^n : \mathcal{N}(\!|\tau|\!)$ are closed computations, $M^v \ltimes^{\diamond}_{\mathcal{M}} M^n$ is equivalent to $[\![M^v]\!]\; \mathcal{R}[\![\tau]\!]_1 \; [\![M^n]\!]$. (This is immediate from Lemma 4.5(1) below.)

As we mention above, our goal is relate the observable behaviour of $\mathcal{V}(\!|e|\!)$ to the observable behaviour of $\mathcal{N}(\!|e|\!)$. Precisely, we want to prove $\mathcal{V}(\!|e|\!) \ltimes^{\Gamma}_{\mathcal{M}} \mathcal{N}(\!|e|\!)$. By considering what this means for specific expressions e, we can see that this is not true in general, for three reasons:

- Consider the expression
$$e = (\lambda x\!:\!\mathbf{bool}.\,()) : \mathbf{bool} \to \mathbf{unit}$$
If we apply this to an argument, then in call-by-value we evaluate the argument but in call-by-name we do not.
- Consider the expression
$$e = (\lambda x\!:\!\mathbf{bool}.\,\mathbf{if}\ x\ \mathbf{then}\ x\ \mathbf{else}\ x) : \mathbf{bool} \to \mathbf{bool}$$
If we apply this, then in call-by-value the argument is evaluated once, but in call-by-name the argument is evaluated twice.
- Consider the expression
$$e = (\lambda x\!:\!\mathbf{bool}.\,\lambda y\!:\!\mathbf{bool}.\,\mathbf{if}\ y\ \mathbf{then}\ x\ \mathbf{else}\ x) : \mathbf{bool} \to \mathbf{bool} \to \mathbf{bool}$$
In call-by-value the argument to the outer function is evaluated first, and the argument to the inner function is evaluated second. In call-by-name the arguments are evaluated in the opposite order.

This suggests we should assume that computations can be discarded, copied, and commuted with other computations. Precisely, we want the following properties.

Definition 4.2. Let T be a strong **Poset**-monad. A morphism $f : X \to TY$ is:

- *lax discardable* when $T\langle\rangle_Y \circ f \sqsubseteq \eta_1 \circ \langle\rangle_X$;
- *lax copyable* when $T\langle id_Y, id_Y\rangle \circ f \sqsubseteq seq^{\mathrm{L}}_{Y,Y} \circ \langle f, f\rangle$, equivalently, when $T\langle id_Y, id_Y\rangle \circ f \sqsubseteq seq^{\mathrm{R}}_{Y,Y} \circ \langle f, f\rangle$;
- *lax central* when $seq^{\mathrm{L}}_{Y,W} \circ (f \times id_{TW}) \sqsubseteq seq^{\mathrm{R}}_{Y,W} \circ (f \times id_{TW})$ for all $W \in |\mathbf{C}|$.

The non-lax versions of these properties were first defined by Führmann [Füh99].

Example 4.3. For each of our examples from Section 3.3, every morphism $f : X \to TY$ is lax discardable, lax copyable, and lax central.

Here we define these three properties for morphisms in the model $\mathcal{M}$, but there are similar notions for CBPV computations, as the following lemma shows.

Lemma 4.4. *Let $\Gamma \vdash_c M : \mathbf{F}\,A$ be a CBPV computation. The following hold for every CBPV model that is adequate with respect to a program relation $\preccurlyeq$.*

- *If $[\![M]\!]$ is lax discardable, then*

$$M \textbf{ to } x.\, \textbf{return}\,() \;\preccurlyeq^{\Gamma}_{\mathrm{ctx}}\; \textbf{return}\,()$$

- *If $[\![M]\!]$ is lax copyable, then*

$$M \textbf{ to } x.\, \textbf{return}\,(x,x) \;\preccurlyeq^{\Gamma}_{\mathrm{ctx}}\; M \textbf{ to } x_1.\, M \textbf{ to } x_2.\, \textbf{return}\,(x_1,x_2)$$

- *If $[\![M]\!]$ is lax central, then*

$$M \textbf{ to } x.\, \textbf{force}\, z \textbf{ to } y.\, \textbf{return}\,(x,y) \;\preccurlyeq^{\Gamma, z:\mathbf{U}(\mathbf{F}B)}_{\mathrm{ctx}}\; \textbf{force}\, z \textbf{ to } y.\, M \textbf{ to } x.\, \textbf{return}\,(x,y)$$

Proof. Since we assume adequacy, in each case we can reason inside the model.

- If $[\![M]\!]$ is lax discardable, then

$$\begin{aligned} [\![M \textbf{ to } x.\, \textbf{return}\,()]\!] &= T\langle\rangle_{[\![A]\!]} \circ [\![M]\!] \\ &\sqsubseteq \eta_1 \circ \langle\rangle_{[\![\Gamma]\!]} \\ &= [\![\textbf{return}\,()]\!] \end{aligned}$$

- If $[\![M]\!]$ is lax copyable, then

$$\begin{aligned} [\![M \textbf{ to } x.\, \textbf{return}\,(x,x)]\!] &= T\langle id_{[\![A]\!]}, id_{[\![A]\!]}\rangle \circ [\![M]\!] \\ &\sqsubseteq seq^{\mathrm{L}}_{[\![A]\!],[\![A]\!]} \circ \langle [\![M]\!], [\![M]\!]\rangle \\ &= [\![M \textbf{ to } x_1.\, M \textbf{ to } x_2.\, \textbf{return}\,(x_1,x_2)]\!] \end{aligned}$$

- If $[\![M]\!]$ is lax central, then

$$\begin{aligned} [\![M \textbf{ to } x.\, \textbf{force}\, z \textbf{ to } y.\, \textbf{return}\,(x,y)]\!] &= seq^{\mathrm{L}}_{[\![A]\!],[\![B]\!]} \circ ([\![M]\!] \times id_{T[\![B]\!]}) \\ &\sqsubseteq seq^{\mathrm{R}}_{[\![A]\!],[\![B]\!]} \circ ([\![M]\!] \times id_{T[\![B]\!]}) \\ &= [\![\textbf{force}\, z \textbf{ to } y.\, M \textbf{ to } x.\, \textbf{return}\,(x,y)]\!] \end{aligned}$$

□

We turn to the proof that lax discardability, lax copyability and lax centrality are sufficient to relate call-by-value to call-by-name. The following two lemmas are useful for this. The first lemma says that (even without assuming these properties of side-effects), the relations $\mathcal{R}[\![\tau]\!]$ are closed under various operations.

Lemma 4.5. *Let $\mathcal{M}$ be a CBPV model. For each τ, the family of relations $\mathcal{R}[\![\tau]\!]$ has the following closure properties.*

(1) *For all f^v, g^v, f^n, g^n, we have*

$$g^v \sqsubseteq f^v \;\wedge\; f^v\; \mathcal{R}[\![\tau]\!]_W\; f^n \;\wedge\; f^n \sqsubseteq g^n \quad\Rightarrow\quad g^v\; \mathcal{R}[\![\tau]\!]_W\; g^n$$

(2) *For all $w : W' \to W$, and f^v, f^n, we have*

$$f^v\; \mathcal{R}[\![\tau]\!]_W\; f^n \quad\Rightarrow\quad (f^v \circ w)\; \mathcal{R}[\![\tau]\!]_{W'}\; (f^n \circ w)$$

(3) *For all f^v, f^n, we have*

$$f^v \ \mathcal{R}[\![\tau]\!]_{W\times X} \ f^n \quad \Rightarrow \quad (f^v)^{\dagger^{W\times\Box}} \ \mathcal{R}[\![\tau]\!]_{W\times TX} \ (f^n)^{\ddagger^{W\times\Box}}$$

(4) *For all W_1, W_2 such that the coproduct $W_1 + W_2$ exists, and all $f_1^v, f_2^v, f_1^n, f_2^n$, we have*

$$f_1^v \ \mathcal{R}[\![\tau]\!]_{W_1} \ f_1^n \ \wedge \ f_2^v \ \mathcal{R}[\![\tau]\!]_{W_2} \ f_2^n \quad \Rightarrow \quad [f_1^v, f_2^v] \ \mathcal{R}[\![\tau]\!]_{W_1+W_2} \ [f_1^n, f_2^n]$$

Proof. The proof of each property is by induction on the type τ.

(1) The **unit** and **bool** cases are trivial, while the cases for product and function types follow from the inductive hypothesis by monotonicity of composition and pairing.

(2) The **unit** and **bool** cases follow from monotonicity of composition. The case for product types is immediate from the inductive hypothesis. For a function type $\tau \to \tau'$ we need to show, for every $w' : W'' \to W'$ and g^v, g^n, that $g^v \ \mathcal{R}[\![\tau]\!]_{W''} \ g^n$ implies

$$(ev^\dagger \circ \langle\!\langle f^v \circ w \circ w', g^v \rangle\!\rangle) \ \mathcal{R}[\![\tau']\!] \ (ev \circ \langle f^n \circ w \circ w', g^n\rangle)$$

This follows immediately from the assumption $f^v \ \mathcal{R}[\![\tau \to \tau']\!]_W \ f^n$, instantiated with the morphism $w \circ w' : W'' \to W$.

(3) The **unit** and **bool** cases follow from monotonicity of extension operators. The case for product types follows from the inductive hypothesis, by naturality of extension operators and the definition of the product T-algebra:

$$(T\pi_i \circ (f^v)^{\dagger^{W\times\Box}}) = ((T\pi_i \circ f^v)^{\dagger^{W\times\Box}}) \ \mathcal{R}[\![\tau_i]\!]_{W\times TX} \ ((\pi_i \circ f^n)^{\ddagger^{W\times\Box}}) = (\pi_i \circ (f^n)^{\ddagger^{W\times\Box}})$$

For a function type $\tau \to \tau'$ we show, for every $w : W' \to W \times TX$ and g^v, g^n satisfying $g^v \ \mathcal{R}[\![\tau]\!]_{W'} \ g^n$, that

$$(ev^\dagger \circ \langle\!\langle (f^v)^{\dagger^{W\times\Box}} \circ w, g^v \rangle\!\rangle) \ \mathcal{R}[\![\tau']\!]_{W'} \ (ev \circ \langle (f^n)^{\ddagger^{W\times\Box}} \circ w, g^n\rangle)$$

By property (2) above, we have

$$(g^v \circ \pi_2) \ \mathcal{R}[\![\tau]\!]_{(W\times X)\times W'} \ (g^n \circ \pi_2)$$

so that $f^v \ \mathcal{R}[\![\tau \to \tau']\!]_{W\times X} \ f^n$ implies

$$(ev^\dagger \circ \langle\!\langle f^v \circ \pi_1, g^v \circ \pi_2 \rangle\!\rangle) \ \mathcal{R}[\![\tau']\!]_{(W\times X)\times W'} \ (ev \circ \langle f^n \circ \pi_1, g^n \circ \pi_2\rangle)$$

Hence, by applying (2) and the inductive hypothesis for τ', we have $h^v \ \mathcal{R}[\![\tau']\!]_{W'} \ h^n$, where we define

$$\begin{aligned} h^v &= (ev^\dagger \circ \langle\!\langle f^v \circ \pi_1, g^v \circ \pi_2 \rangle\!\rangle \circ \beta_{W,W',X})^{\dagger^{(W\times W')\times\Box}} \circ \beta_{W,TX,W'} \circ \langle w, id_{W'}\rangle \\ h^n &= (ev \circ \langle f^n \circ \pi_1, g^n \circ \pi_2\rangle \circ \beta_{W,W',X})^{\ddagger^{(W\times W')\times\Box}} \circ \beta_{W,TX,W'} \circ \langle w, id_{W'}\rangle \end{aligned}$$

with β as in Definition 3.5. It then remains to show that h^v and h^n are the two sides of the required instance of $\mathcal{R}[\![\tau']\!]_{W'}$, which we prove as follows. To prove we have the correct left-hand side, we use the associativity law, naturality of Kleisli extension, and the associativity law again, as follows.

$$\begin{aligned} h^v &= ev^\dagger \circ (\langle\!\langle f^v \circ \pi_1, g^v \circ \pi_2 \rangle\!\rangle \circ \beta_{W,W',X})^{\dagger^{(W\times W')\times\Box}} \circ \beta_{W,TX,W'} \circ \langle w, id_{W'}\rangle \\ &= ev^\dagger \circ (seq^{\mathrm{L}} \circ (f^v \times id) \circ \beta_{W,T(\mathcal{V}[\![\tau]\!]),X})^{\dagger^{(W\times T(\mathcal{V}[\![\tau]\!]))\times\Box}} \circ \beta_{W,TX,T(\mathcal{V}[\![\tau]\!])} \circ \langle w, g^v\rangle \\ &= ev^\dagger \circ seq^{\mathrm{L}} \circ ((f^v)^{\dagger^{W\times\Box}} \times id_{T(\mathcal{V}[\![\tau]\!])}) \circ \langle w, g^v\rangle \\ &= ev^\dagger \circ \langle\!\langle (f^v)^{\dagger^{W\times\Box}} \circ w, g^v \rangle\!\rangle \end{aligned}$$

To prove we have the correct right-hand side, we use naturality of extension, and the definition of power T-algebras, as follows.

$$\begin{aligned} h^n &= (ev \circ (f^n \times id_{U_{\mathsf{T}}(\mathcal{N}[\![\tau]\!])}) \circ \beta_{W,U_{\mathsf{T}}(\mathcal{N}[\![\tau]\!]),X})^{\ddagger^{(W\times W')\times\Box}} \circ \beta_{W,TX,U_{\mathsf{T}}(\mathcal{N}[\![\tau]\!])} \circ \langle w, g^n\rangle \\ &= ev \circ ((f^n)^{\ddagger^{W\times\Box}} \times id_{U_{\mathsf{T}}(\mathcal{N}[\![\tau]\!])}) \circ \langle w, g^n\rangle \\ &= ev \circ \langle (f^n)^{\ddagger^{W\times\Box}} \circ w, g^n\rangle \end{aligned}$$

(4) The **unit** and **bool** cases are immediate from monotonicity of the copairing operator $[-,-]$. For product types, it is enough to note that $T\pi_i \circ [f_1^v, f_2^v] = [T\pi_i \circ f_1^v, T\pi_i \circ f_2^v]$ and $\pi_i \circ [f_1^n, f_2^n] = [\pi_i \circ f_1^n, \pi_i \circ f_2^n]$, and then apply the inductive hypothesis. For a function type $\tau \to \tau'$, we show that $g^v \; \mathcal{R}[\![\tau]\!]_{W'} \; g^n$ implies

$$(ev^\dagger \circ \langle\!\langle [f_1^v, f_2^v] \circ w, g^v \rangle\!\rangle) \; \mathcal{R}[\![\tau']\!]_{W'} \; (ev \circ \langle [f_1^n, f_2^n] \circ w, g^n\rangle)$$

where $w : W' \to W_1 + W_2$. To do this, consider the following objects W_1', W_2' and morphisms w_1', w_2'.

$$W_i' = W' \times W_i \qquad w_i' = \pi_2 : W_i' \to W_i \qquad (i \in \{1,2\})$$

Property (2) implies $(g^v \circ \pi_1) \; \mathcal{R}[\![\tau]\!]_{W_i'} \; (g^n \circ \pi_1)$, so from the assumption $f_i^v \; \mathcal{R}[\![\tau \to \tau']\!]_{W_i} \; f_i^n$, we obtain $k_i^v \; \mathcal{R}[\![\tau']\!]_{W_i'} \; k_i^n$, where

$$k_i^v = ev^\dagger \circ \langle\!\langle f_i^v \circ w_i', g^v \circ \pi_1 \rangle\!\rangle \qquad k_i^n = ev \circ \langle f_i^n \circ w_i', g^n \circ \pi_1\rangle$$

Coproducts are distributive (because we assume cartesian closure), so the coproduct $W_1' + W_2'$ exists and is isomorphic to $W' \times (W_1 + W_2)$. We can therefore apply the inductive hypothesis for τ, and then (2), to obtain

$$([k_1^v, k_2^v] \circ h) \; \mathcal{R}[\![\tau']\!]_{W'} \; ([k_1^n, k_2^n] \circ h)$$

where

$$h : W' \xrightarrow{\langle id_{W'}, w\rangle} W' \times (W_1 + W_2) \xrightarrow{\cong} W_1' + W_2'$$

The result follows because

$$[k_1^v, k_2^v] \circ h = ev^\dagger \circ \langle\!\langle [f_1^v, f_2^v] \circ w, g^v \rangle\!\rangle \qquad [k_1^n, k_2^n] \circ h = ev \circ \langle [f_1^n, f_2^n] \circ w, g^n\rangle \qquad \square$$

The second lemma consists of useful consequences of lax discardability, lax copyability and lax centrality.

Lemma 4.6. *Let T be a strong* **Poset***-monad.*

(1) *Let $f_1 : W \to TX_1$ and $f_2 : W \to TX_2$ be morphisms. For each $i \in \{1,2\}$, if f_j is lax discardable for $j \neq i$, then*

$$T\pi_i \circ \langle\!\langle f_1, f_2 \rangle\!\rangle \;\sqsubseteq\; f_i$$

(2) *Let $f_1 : W_1 \to TX_1$, $f_2 : W_2 \to TX_2$ and $g : X_1 \times X_2 \to TY$ be morphisms. If f_1 is lax central, then*

$$(g^{\dagger^{X_1\times\Box}})^{\dagger^{\Box\times TX_2}} \circ (f_1 \times f_2) \;\sqsubseteq\; (g^{\dagger^{\Box\times X_2}})^{\dagger^{TX_1\times\Box}} \circ (f_1 \times f_2)$$

If f_2 is lax central, then

$$(g^{\dagger^{\Box\times X_2}})^{\dagger^{TX_1\times\Box}} \circ (f_1 \times f_2) \;\sqsubseteq\; (g^{\dagger^{X_1\times\Box}})^{\dagger^{\Box\times TX_2}} \circ (f_1 \times f_2)$$

(3) *Let $f : W \to TX$, $g : X \to TY$ and $h : X \times Y \to TZ$ be morphisms. If f is lax copyable and lax central, then*

$$(h^{\dagger^{X\times\Box}} \circ \langle id_X, g\rangle)^\dagger \circ f \;\sqsubseteq\; (h^{\dagger^{\Box\times Y}} \circ (f \times id_Y))^{\dagger^{W\times\Box}} \circ \langle id_W, g^\dagger \circ f\rangle$$

(4) *Let $f : W \to TX$, $g_1 : X \to TY_1$ and $g_2 : X \to TY_2$ be morphisms. If f is lax copyable and lax central, then*

$$\langle\!\langle g_1, g_2\rangle\!\rangle^\dagger \circ f \;\sqsubseteq\; \langle\!\langle {g_1}^\dagger \circ f, {g_2}^\dagger \circ f\rangle\!\rangle$$

Proof. (1) The following proves the statement for $i = 1$; the proof for $i = 2$ is similar.

$$\begin{aligned}
T\pi_1 \circ \langle\!\langle f_1, f_2\rangle\!\rangle &= T\pi_1 \circ T(id_{X_1} \times \langle\rangle_{X_2}) \circ \langle\!\langle f_1, f_2\rangle\!\rangle \\
&= T\pi_1 \circ \langle\!\langle f_1, T\langle\rangle_{X_2} \circ f_2\rangle\!\rangle && \text{(naturality of extension)} \\
&\sqsubseteq T\pi_1 \circ \langle\!\langle f_1, \eta_1 \circ \langle\rangle_W\rangle\!\rangle && \text{(lax discardability of } f_2\text{)} \\
&= T\pi_1 \circ (\eta_{X_1\times 1})^{\dagger^{\Box\times 1}} \circ \langle f_1, \langle\rangle_W\rangle && \text{(left unit law)} \\
&= f_1 && \text{(right unit law)}
\end{aligned}$$

(2) Lax centrality of f_1 implies the first inequality of (2) as follows.

$$\begin{aligned}
&(g^{\dagger^{X_1\times\Box}})^{\dagger^{\Box\times TX_2}} \circ (f_1 \times f_2) \\
&= g^\dagger \circ seq^{\mathrm{L}}_{X_1,X_2} \circ (f_1 \times id_{TX_2}) \circ (id_{W_1} \times f_2) && \text{(associativity and left unit laws)} \\
&\sqsubseteq g^\dagger \circ seq^{\mathrm{R}}_{X_1,X_2} \circ (f_1 \times id_{TX_2}) \circ (id_{W_1} \times f_2) && \text{(lax centrality)} \\
&= (g^{\dagger^{\Box\times X_2}})^{\dagger^{TX_1\times\Box}} \circ (f_1 \times f_2) && \text{(associativity and left unit laws)}
\end{aligned}$$

For the other inequality, precomposing with the isomorphisms $\langle\pi_2, \pi_1\rangle$ swaps the roles of f_1 and f_2, so that we can reuse the first inequality.

(3) The following proves the result.

$$\begin{aligned}
&(h^{\dagger^{X\times\Box}} \circ \langle id_X, g\rangle)^\dagger \circ f \\
&= (h^{\dagger^{X\times\Box}} \circ (id_X \times g))^\dagger \circ T\langle id_X, id_X\rangle \circ f && \text{(naturality of extension)} \\
&\sqsubseteq (h^{\dagger^{X\times\Box}} \circ (id_X \times g))^\dagger \circ seq^{\mathrm{R}}_{X,X} \circ \langle f, f\rangle && \text{(lax copyability)} \\
&= ((h^{\dagger^{X\times\Box}} \circ (id_X \times g))^{\dagger^{\Box\times X}})^{\dagger^{TX\times\Box}} \circ \langle f, f\rangle && \text{(associativity, left unit)} \\
&= ((h^{\dagger^{X\times\Box}})^{\dagger^{\Box\times TY}} \circ (f \times g))^{\dagger^{W\times\Box}} \circ \langle id_W, f\rangle && \text{(naturality of extension)} \\
&\sqsubseteq ((h^{\dagger^{\Box\times Y}})^{\dagger^{TX\times\Box}} \circ (f \times g))^{\dagger^{W\times\Box}} \circ \langle id_W, f\rangle && \text{(Lemma 4.6(2))} \\
&= ((h^{\dagger^{\Box\times Y}} \circ (f \times id_Y))^{\dagger^{W\times\Box}} \circ (id_W \times g))^{\dagger^{W\times\Box}} \circ \langle id_W, f\rangle && \text{(naturality of extension)} \\
&= (h^{\dagger^{\Box\times Y}} \circ (f \times id_Y))^{\dagger^{W\times\Box}} \circ (id_W \times g^\dagger) \circ \langle id_W, f\rangle && \text{(associativity law)} \\
&= (h^{\dagger^{\Box\times Y}} \circ (f \times id_Y))^{\dagger^{W\times\Box}} \circ \langle id_W, g^\dagger \circ f\rangle
\end{aligned}$$

(4) The required inequality is equivalent, by postcomposing with the isomorphism $T\langle \pi_2, \pi_1 \rangle$, to

$$(seq^{\mathrm{R}}_{Y_2,Y_1} \circ \langle g_2, g_1 \rangle)^\dagger \circ f \;\sqsubseteq\; seq^{\mathrm{R}}_{Y_2,Y_1} \circ \langle {g_2}^\dagger \circ f, {g_1}^\dagger \circ f \rangle$$

We prove this as follows, using (3) with $h = {\eta_{Y_2 \times Y_1}}^{\dagger^{\Box \times Y_1}} \circ (g_2 \times id_{Y_1})$.

$$\begin{aligned}
&(seq^{\mathrm{R}}_{Y_2,Y_1} \circ \langle g_2, g_1 \rangle)^\dagger \circ f \\
&= ((\eta^{\dagger^{\Box \times Y_1}} \circ (g_2 \times id_{Y_1}))^{\dagger^{X \times \Box}} \circ \langle id_X, g_1 \rangle)^\dagger \circ f && \text{(naturality of extension)} \\
&\sqsubseteq ((\eta^{\dagger^{\Box \times Y_1}} \circ (g_2 \times id_{Y_1}))^{\dagger^{\Box \times Y_1}} \circ (f \times id_{Y_1}))^{\dagger^{W \times \Box}} \circ \langle id_W, {g_1}^\dagger \circ f \rangle && \text{(Lemma 4.6(3))} \\
&= (\eta^{\dagger^{\Box \times Y_1}} \circ ({g_2}^\dagger \times id_{Y_1}) \circ (f \times id_{Y_1}))^{\dagger^{W \times \Box}} \circ \langle id_W, {g_1}^\dagger \circ f \rangle && \text{(associativity law)} \\
&= seq^{\mathrm{R}}_{Y_2,Y_1} \circ \langle {g_2}^\dagger \circ f, {g_1}^\dagger \circ f \rangle && \text{(naturality of extension)}
\end{aligned}$$

□

We are now ready to prove the main result of this section.

Theorem 4.7. *Let $\mathcal{M} = (\mathbf{C}, \mathsf{T})$ be a CBPV model. If every morphism $f : X \to TY$ is lax discardable, lax copyable, and lax central, then for every expression $\Gamma \vdash e : \tau$ we have*

$$\mathcal{V}(\!|e|\!) \;\ltimes^{\Gamma}_{\mathcal{M}}\; \mathcal{N}(\!|e|\!)$$

Proof. We prove, by induction on the expression e, that for all families of morphisms

$$(f^v_i : W \to T(\mathcal{V}[\![\tau_i]\!]))_i \qquad (f^n_i : W \to U_{\mathsf{T}}(\mathcal{N}[\![\tau_i]\!]))_i$$

we have

$$f^v_1 \;\mathcal{R}[\![\tau_1]\!]_W\; f^n_1 \;\wedge\; \cdots \;\wedge\; f^v_k \;\mathcal{R}[\![\tau_k]\!]_W\; f^n_k \quad\Rightarrow\quad (\mathcal{V}[\![e]\!]^\dagger \circ \langle\!\langle f^v_i \rangle\!\rangle_i) \;\mathcal{R}[\![\tau']\!]_W\; (\mathcal{N}[\![e]\!] \circ \langle f^n_i \rangle_i)$$

- For a variable x_j, we are required to prove

  $$(T\pi_j \circ \langle\!\langle f^v_i \rangle\!\rangle_i) \;\mathcal{R}[\![\tau_j]\!]_W\; f^n_i$$

  A simple induction on j, using Lemma 4.6(1), tells us that

  $$T\pi_j \circ \langle\!\langle f^v_i \rangle\!\rangle_i \;\sqsubseteq\; f^v_j$$

  The result then follows from the assumption $f^v_j \;\mathcal{R}[\![\tau_j]\!]_W\; f^n_j$ via Lemma 4.5(1).
- For the expression (), we are required to show

  $$(T\langle\rangle_{\mathcal{V}[\![\Gamma]\!]} \circ \langle\!\langle f^v_i \rangle\!\rangle_i) \;\mathcal{R}[\![\mathbf{unit}]\!]_W\; (\eta_1 \circ \langle\rangle_W)$$

  By definition of $\mathcal{R}[\![\mathbf{unit}]\!]$, this is the same as

  $$(T\langle\rangle_{\mathcal{V}[\![\Gamma]\!]}) \circ \langle\!\langle f^v_i \rangle\!\rangle_i) \sqsubseteq (\eta_1 \circ \langle\rangle_W)$$

  which is immediate from lax discardability.
- For a pair (e_1, e_2), the inductive hypothesis tells us that

  $$(\mathcal{V}[\![e_j]\!]^\dagger \circ \langle\!\langle f^v_i \rangle\!\rangle_i) \;\mathcal{R}[\![\tau'_j]\!]_W\; (\mathcal{N}[\![e_j]\!] \circ \langle f^n_i \rangle_i)$$

  for each $j \in \{1, 2\}$. We also have

  $$\begin{aligned}
  &T\pi_j \circ \langle\!\langle \mathcal{V}[\![e_1]\!], \mathcal{V}[\![e_2]\!] \rangle\!\rangle^\dagger \circ \langle\!\langle f^v_i \rangle\!\rangle_i \\
  &\sqsubseteq T\pi_j \circ \langle\!\langle (\mathcal{V}[\![e_1]\!]^\dagger \circ \langle\!\langle f^v_i \rangle\!\rangle_i), (\mathcal{V}[\![e_2]\!]^\dagger \circ \langle\!\langle f^v_i \rangle\!\rangle_i) \rangle\!\rangle && \text{(Lemma 4.6(4))} \\
  &\sqsubseteq \mathcal{V}[\![e_j]\!]^\dagger \circ \langle\!\langle f^v_i \rangle\!\rangle_i && \text{(Lemma 4.6(1))}
  \end{aligned}$$

so that Lemma 4.5 implies

$$(T\pi_j \circ \langle\!\langle \mathcal{V}[\![e_1]\!], \mathcal{V}[\![e_2]\!] \rangle\!\rangle^\dagger \circ \langle\!\langle f_i^v \rangle\!\rangle_i) \quad \mathcal{R}[\![\tau_j']\!]_W \quad (\pi_j \circ \langle \mathcal{N}[\![e_1]\!], \mathcal{N}[\![e_2]\!] \rangle \circ \langle f_i^n \rangle_i)$$

Hence

$$(\langle\!\langle \mathcal{V}[\![e_1]\!], \mathcal{V}[\![e_2]\!] \rangle\!\rangle^\dagger \circ \langle\!\langle f_i^v \rangle\!\rangle_i)\ \mathcal{R}[\![\tau_1' \times \tau_2']\!]_W\ (\langle \mathcal{N}[\![e_1]\!], \mathcal{N}[\![e_2]\!] \rangle \circ \langle f_i^n \rangle_i)$$

as required.

- For **fst** e, we need to show

$$(T\pi_1 \circ \mathcal{V}[\![e]\!]^\dagger \circ \langle\!\langle f_i^v \rangle\!\rangle_i)\ \mathcal{R}[\![\tau_1']\!]_W\ (\pi_1 \circ \mathcal{N}[\![e]\!] \circ \langle f_i^n \rangle_i)$$

which is immediate from the inductive hypothesis and the definition of $\mathcal{R}[\![\tau_1' \times \tau_2']\!]$.
- The **snd** case is similar to the **fst** case.
- For the expression **true**, we need to show

$$(T(inl \circ \langle\rangle_{\mathcal{V}[\![\Gamma]\!]}) \circ \langle\!\langle f_i^v \rangle\!\rangle_i)\ \mathcal{R}[\![\mathbf{bool}]\!]_W\ (\eta_2 \circ inl \circ \langle\rangle_W)$$

which means

$$(T(inl \circ \langle\rangle_{\mathcal{V}[\![\Gamma]\!]}) \circ \langle\!\langle f_i^v \rangle\!\rangle_i)\ \sqsubseteq\ (\eta_2 \circ inl \circ \langle\rangle_W)$$

This follows from lax discardability, and naturality of η.
- The **false** case is similar to the **true** case.
- For **if** e_0 **then** e_1 **else** e_2, the inductive hypothesis gives us

$$g_0^v \sqsubseteq g_0^n \qquad g_1^v\ \mathcal{R}[\![\tau']\!]_W\ g_1^n \qquad g_2^v\ \mathcal{R}[\![\tau']\!]_W\ g_2^n$$

where we define

$$g_j^v = \mathcal{V}[\![e_j]\!]^\dagger \circ \langle\!\langle f_i^v \rangle\!\rangle_i \qquad g_j^n = \mathcal{N}[\![e_j]\!] \circ \langle f_i^n \rangle_i \qquad (j \in \{0,1,2\})$$

By applying all of the closure properties of Lemma 4.5, we therefore have

$$(([g_1^v, g_2^v] \circ dist)^{\dagger^{W \times \Box}} \circ (id_W \times g_0^v) \circ \langle id, id \rangle)\ \mathcal{R}[\![\tau']\!]_W\ (([g_1^n, g_2^n] \circ dist)^{\ddagger^{W \times \Box}} \circ (id_W \times g_0^n) \circ \langle id, id \rangle)$$

This is not quite what we need, but it does imply the result via another use of Lemma 4.5(1), as follows. We have

$$([\mathcal{V}[\![e_1]\!], \mathcal{V}[\![e_2]\!]] \circ dist)^{\dagger^{\Box \times 2}} = [\mathcal{V}[\![e_1]\!]^\dagger, \mathcal{V}[\![e_2]\!]^\dagger] \circ dist$$

by precomposing with $dist^{-1}$ and using the universal property of the coproduct $T(\mathcal{V}[\![\Gamma]\!]) + T(\mathcal{V}[\![\Gamma]\!])$. It follows that

$$\begin{aligned}
&\mathcal{V}[\![\mathbf{if}\ e_0\ \mathbf{then}\ e_1\ \mathbf{else}\ e_2]\!]^\dagger \circ \langle\!\langle f_i^v \rangle\!\rangle \\
&= (([\mathcal{V}[\![e_1]\!], \mathcal{V}[\![e_2]\!]] \circ dist)^{\dagger^{\mathcal{V}[\![\Gamma]\!] \times \Box}} \circ \langle id, \mathcal{V}[\![e_0]\!] \rangle)^\dagger \circ \langle\!\langle f_i^v \rangle\!\rangle_i \\
&\sqsubseteq (([\mathcal{V}[\![e_1]\!], \mathcal{V}[\![e_2]\!]] \circ dist)^{\dagger^{\Box \times 2}} \circ (\langle\!\langle f_i^v \rangle\!\rangle_i \times id_2))^{\dagger^{W \times \Box}} \circ \langle id_W, g_0^v \rangle \qquad \text{(Lemma 4.6(3))} \\
&= ([\mathcal{V}[\![e_1]\!]^\dagger, \mathcal{V}[\![e_2]\!]^\dagger] \circ dist \circ (\langle\!\langle f_i^v \rangle\!\rangle_i \times id_2))^{\dagger^{W \times \Box}} \circ \langle id_W, g_0^v \rangle \\
&= ([g_1^v, g_2^v] \circ dist)^{\dagger^{W \times \Box}} \circ (id_W \times g_0^v) \circ \langle id, id \rangle
\end{aligned}$$

We also have

$$\mathcal{N}[\![\mathbf{if}\ e_0\ \mathbf{then}\ e_1\ \mathbf{else}\ e_2]\!] \circ \langle f_i^n \rangle_i\ =\ ([g_1^n, g_2^n] \circ dist)^{\ddagger^{W \times \Box}} \circ (id_W \times g_0^n) \circ \langle id, id \rangle$$

so that Lemma 4.5(1) implies

$$(\mathcal{V}[\![\mathbf{if}\ e_0\ \mathbf{then}\ e_1\ \mathbf{else}\ e_2]\!]^\dagger \circ \langle\!\langle f_i^v \rangle\!\rangle)\ \mathcal{R}[\![\tau']\!]_W\ (\mathcal{N}[\![\mathbf{if}\ e_0\ \mathbf{then}\ e_1\ \mathbf{else}\ e_2]\!] \circ \langle f_i^n \rangle_i)$$

as required.

- For a λ-abstraction $\lambda x\!:\!\tau.\, e$ of type $\tau \to \tau'$, consider arbitrary $w : W' \to W$ and g^v, g^n such that
$$g^v \ \mathcal{R}[\![\tau]\!]_{W'} \ g^n$$
We need to show
$$(ev^\dagger \circ \langle\!\langle (\mathcal{V}[\![\lambda x\!:\!\tau.\, e]\!]^\dagger \circ \langle\!\langle f_i^v \rangle\!\rangle_i \circ w), g^v \rangle\!\rangle) \ \mathcal{R}[\![\tau']\!]_{W'} \ (ev \circ \langle \mathcal{N}[\![\lambda x\!:\!\tau.\, e]\!] \circ \langle f_i^n \rangle_i \circ w, g^n \rangle)$$
Applying Lemma 4.5(3) to the assumptions yields
$$(f_i^v \circ w) \ \mathcal{R}[\![\tau_i]\!]_{W'} \ (f_i^n \circ w)$$
so that, by the inductive hypothesis, we have
$$(\mathcal{V}[\![e]\!]^\dagger \circ \langle\!\langle \langle\!\langle f_i^v \circ w \rangle\!\rangle_i, g^v \rangle\!\rangle) \ \mathcal{R}[\![\tau']\!]_{W'} \ (\mathcal{N}[\![e]\!] \circ \langle \langle f_i^n \circ w \rangle_i, g^n \rangle)$$
We show that this is the required instance of $\mathcal{R}[\![\tau']\!]_{W'}$ by rewriting both sides as follows. For the left-hand side, we have
$$\begin{aligned}
& ev^\dagger \circ \langle\!\langle (\mathcal{V}[\![\lambda x\!:\!\tau.\, e]\!]^\dagger \circ \langle\!\langle f_i^v \rangle\!\rangle_i \circ w), g^v \rangle\!\rangle \\
= \ & ev^\dagger \circ \langle\!\langle T(\Lambda(\mathcal{V}[\![e]\!])) \circ \langle\!\langle f_i^v \circ w \rangle\!\rangle_i, g^v \rangle\!\rangle \\
= \ & (ev \circ (\Lambda(\mathcal{V}[\![e]\!]) \times id_{\mathcal{V}[\![\tau]\!]}))^\dagger \circ \langle\!\langle \langle\!\langle f_i^v \circ w \rangle\!\rangle_i, g^v \rangle\!\rangle \quad \text{(naturality of extension)} \\
= \ & \mathcal{V}[\![e]\!]^\dagger \circ \langle\!\langle \langle\!\langle f_i^v \circ w \rangle\!\rangle_i, g^v \rangle\!\rangle
\end{aligned}$$
For the right-hand side:
$$\begin{aligned}
ev \circ \langle \mathcal{N}[\![\lambda x\!:\!\tau.\, e]\!] \circ \langle f_i^n \rangle_i \circ w, g^n \rangle \ &= \ ev \circ \langle \Lambda(\mathcal{N}[\![e]\!]) \circ \langle f_i^n \circ w \rangle_i, g^n \rangle \\
&= \mathcal{N}[\![e]\!] \circ \langle \langle f_i^n \circ w \rangle_i, g^n \rangle
\end{aligned}$$
- For an application $e\, e'$, where e has type $\tau \to \tau$, the inductive hypothesis for e' gives us
$$(\mathcal{V}[\![e']\!]^\dagger \circ \langle\!\langle f_i^v \rangle\!\rangle_i) \ \mathcal{R}[\![\tau]\!]_W \ (\mathcal{N}[\![e']\!] \circ \langle f_i^n \rangle_i)$$
so that, by the inductive hypothesis for e with $w = id_W$, we have
$$(ev^\dagger \circ \langle\!\langle (\mathcal{V}[\![e]\!]^\dagger \circ \langle\!\langle f_i^v \rangle\!\rangle_i), (\mathcal{V}[\![e']\!]^\dagger \circ \langle\!\langle f_i^v \rangle\!\rangle_i) \rangle\!\rangle) \ \mathcal{R}[\![\tau']\!]_W \ (ev \circ \langle (\mathcal{N}[\![e]\!] \circ \langle f_i^n \rangle_i), (\mathcal{N}[\![e']\!] \circ \langle f_i^n \rangle_i) \rangle)$$
We rewrite both sides as follows. For the left we have
$$\begin{aligned}
& \mathcal{V}[\![e\, e']\!]^\dagger \circ \langle\!\langle f_i^v \rangle\!\rangle_i \\
= \ & (ev^{\dagger^{\mathcal{V}[\![\tau \to \tau']\!] \times \square}})^{\dagger^{\square \times T(\mathcal{V}[\![\tau]\!])}} \circ \langle \mathcal{V}[\![e]\!], \mathcal{V}[\![e']\!] \rangle \circ \langle\!\langle f_i^v \rangle\!\rangle \qquad \text{(naturality of extension)} \\
= \ & ev^\dagger \circ \langle\!\langle \mathcal{V}[\![e]\!], \mathcal{V}[\![e']\!] \rangle\!\rangle^\dagger \circ \langle\!\langle f_i^v \rangle\!\rangle \qquad \text{(left unit, associativity)} \\
= \ & ev^\dagger \circ \langle\!\langle (\mathcal{V}[\![e]\!]^\dagger \circ \langle\!\langle f_i^v \rangle\!\rangle_i), (\mathcal{V}[\![e']\!]^\dagger \circ \langle\!\langle f_i^v \rangle\!\rangle_i) \rangle\!\rangle \qquad \text{(Lemma 4.6(4))}
\end{aligned}$$
and for the right,
$$\mathcal{N}[\![e\, e']\!] \circ \langle f_i^n \rangle_i \ = \ ev \circ \langle (\mathcal{N}[\![e]\!] \circ \langle f_i^n \rangle_i), (\mathcal{N}[\![e']\!] \circ \langle f_i^n \rangle_i) \rangle$$
We therefore have
$$(\mathcal{V}[\![e\, e']\!]^\dagger \circ \langle\!\langle f_i^v \rangle\!\rangle_i) \ \mathcal{R}[\![\tau']\!]_W \ (\mathcal{N}[\![e\, e']\!] \circ \langle f_i^n \rangle_i)$$
as required. □

As a corollary, we can directly compare the call-by-value and call-by-name translations of source-language *programs* (closed expressions of type **bool**).

Corollary 4.8. *Let $\mathcal{M} = (\mathbf{C}, \mathsf{T})$ be a CBPV model that is adequate with respect a program relation $\preccurlyeq$. If every morphism $f : X \to TY$ is lax discardable, lax copyable, and lax central, then for every closed expression e : **bool**, we have*

$$\mathcal{V}(\!|e|\!) \preccurlyeq \mathcal{N}(\!|e|\!)$$

Proof. By Theorem 4.7 we have $\mathcal{V}[\![e]\!] \ltimes^{\diamond}_{\mathcal{M}} \mathcal{N}[\![e]\!]$, so in particular $\mathcal{V}[\![e]\!]\ \mathcal{R}[\![\mathbf{bool}]\!]_1\ \mathcal{N}[\![e]\!]$ By definition, the latter means $\mathcal{V}[\![e]\!] \sqsubseteq \mathcal{N}[\![e]\!]$, which implies the result by adequacy. □

4.1. **Examples.** To conclude this section, we discuss the consequences of the results above for each of our examples.

We first note that we can in fact simplify the definition of $\ltimes$ for each of these examples, by using the fact that, in each of the three **Poset**-categories **Set**, **Poset**, ω**Cpo**, morphisms are in particular functions, and are ordered pointwise. (We treat a set as a discrete poset here.) It follows that instead of considering generalized elements $f : W \to X$, it is enough to consider ordinary elements $t \in X$ (which we can identify with morphisms $t : 1 \to X$). The simplification of $\mathcal{R}[\![-]\!]$ we obtain is defined as follows.

$\boxed{t^v\ \mathcal{R}'[\![\tau]\!]\ t^n \text{ (where } t^v \in T(\mathcal{V}[\![\tau]\!]) \text{ and } t^n \in U_{\mathsf{T}}(\mathcal{N}[\![\tau]\!]))}$

$$\begin{array}{rcl}
t^v\ \mathcal{R}'[\![\mathbf{unit}]\!]\ t^n & \text{iff} & t^v \sqsubseteq t^n \\
t^v\ \mathcal{R}'[\![\tau_1 \times \tau_2]\!]\ t^n & \text{iff} & T\pi_1 t^v\ \mathcal{R}'[\![\tau_1]\!]\ \pi_1 t^n\ \wedge\ T\pi_2 t^v\ \mathcal{R}'[\![\tau_2]\!]\ \pi_2 t^n \\
t^v\ \mathcal{R}'[\![\mathbf{bool}]\!]\ t^n & \text{iff} & t^v \sqsubseteq t^n \\
t^v\ \mathcal{R}'[\![\tau \to \tau']\!]\ t^n & \text{iff} & \forall a^v, a^n.\ a^v \mathcal{R}'[\![\tau]\!] a^n\ \Rightarrow\ ev^{\dagger}(seq^{\mathrm{L}}(t^v, a^v))\ \mathcal{R}'[\![\tau']\!]\ t^n a^n
\end{array}$$

The precise relationship between $\mathcal{R}[\![-]\!]$ and $\mathcal{R}'[\![-]\!]$ is as follows.

Lemma 4.9. *For each of our four example models, we have*

$$f^v\ \mathcal{R}[\![\tau]\!]_W\ f^n \quad \Leftrightarrow \quad \forall w \in W.\ f^v w\ \mathcal{R}'[\![\tau]\!]\ f^n w$$

for every $f^v : W \to T(\mathcal{V}[\![\tau]\!])$ and $f^n : W \to U_{\mathsf{T}}(\mathcal{N}[\![\tau]\!])$. Hence, if $\mathcal{V}(\!|\Gamma|\!) \vdash_c M^v : \mathbf{F}\,(\mathcal{V}(\!|\tau'|\!))$ and $\mathcal{N}(\!|\Gamma|\!) \vdash_c M^n : \mathcal{N}(\!|\tau'|\!)$, then we have

$$M^v\ \ltimes^{\Gamma}_{\mathcal{M}}\ M^n$$

exactly when, for all $(a^v_i \in T(\mathcal{V}[\![\tau_i]\!]))_i$ and $(a^n_i \in U_{\mathsf{T}}(\mathcal{N}[\![\tau_i]\!]))_i$,

$$a^v_1\, \mathcal{R}'[\![\tau_1]\!]\, a^n_1\ \wedge\ \cdots\ \wedge\ a^v_k\, \mathcal{R}'[\![\tau_k]\!]\, a^n_k \quad \Rightarrow \quad ([\![M^v]\!]^{\dagger}\langle\!\langle a^v_i \rangle\!\rangle_i)\ \mathcal{R}'[\![\tau']\!]\ ([\![M^n]\!](a^n_i)_i)$$

where $\Gamma = x_1 : \tau_1, \ldots, x_k : \tau_k$.

Proof. By induction on τ. This is trivial for **unit**, **bool** and product types. For a function type $\tau \to \tau'$, the ($\Leftarrow$) direction is again trivial. The ($\Rightarrow$) direction follows from the fact that, identifying an element $w \in W$ with a morphism $w : 1 \to W$, we have $\mathcal{R}'[\![\tau]\!] = \mathcal{R}[\![\tau]\!]_1$ and $\mathcal{R}'[\![\tau']\!] = \mathcal{R}[\![\tau']\!]_1$ by the inductive hypothesis. □

We now consider each of our examples in turn. Note that, since the proof of Theorem 4.7 is by induction on the expression e, we need to extend the proof with cases for the extra syntax we add in these examples.

Corollary 4.10. *For our no side-effects example, we have*

$$\mathcal{V}(\!|e|\!) \ltimes^{\Gamma}_{\mathcal{M}} \mathcal{N}(\!|e|\!)$$

for every $\Gamma \vdash e : \tau$*, and in particular,*

$$\exists V : \mathbf{bool}.\ (\mathcal{V}(\!|e'|\!) \Downarrow \mathbf{return}\, V) \ \wedge\ (\mathcal{N}(\!|e'|\!) \Downarrow \mathbf{return}\, V)$$

for every closed expression $e' : \mathbf{bool}$*.*

Proof. This is immediate from Theorem 4.7 and Corollary 4.8, defining $\preccurlyeq$ as in Example 2.2. □

Corollary 4.11. *For our divergence example, we have*

$$\mathcal{V}(\!|e|\!) \ltimes^{\Gamma}_{\mathcal{M}} \mathcal{N}(\!|e|\!)$$

for every $\Gamma \vdash e : \tau$*, and in particular,*

$$\forall V : \mathbf{bool}.\ (\mathcal{V}(\!|e'|\!) \Downarrow \mathbf{return}\, V) \ \Rightarrow\ (\mathcal{N}(\!|e'|\!) \Downarrow \mathbf{return}\, V)$$

for every closed expression $e' : \mathbf{bool}$*.*

Proof. We first extend the inductive proof of Theorem 4.7 with a case for recursive functions $\mathbf{rec}\, f : \tau \to \tau'.\, \lambda x.\, e$. In light of Lemma 4.9 above, it suffices for this to show that, if

$$h^v : \mathcal{V}[\![\tau \to \tau']\!] \times \mathcal{V}[\![\tau]\!] \to T(\mathcal{V}[\![\tau']\!]) \qquad h^n : U_{\mathsf{T}}(\mathcal{N}[\![\tau \to \tau']\!]) \times U_{\mathsf{T}}(\mathcal{N}[\![\tau]\!]) \to U_{\mathsf{T}}(\mathcal{N}[\![\tau']\!])$$

are ω-continuous functions that satisfy

$$t^v\ \mathcal{R}'[\![\tau \to \tau']\!]\ t^n \ \wedge\ a^v\ \mathcal{R}'[\![\tau]\!]\ a^n \ \Rightarrow\ h^v(t^v, a^v)\ \mathcal{R}'[\![\tau']\!]\ h^n(t^n, a^n)$$

when $t^v, a^v \neq \bot$, then we have

$$\mathit{fix}(\lambda g^v.\, h^v(g^v, -))\ \mathcal{R}'[\![\tau \to \tau']\!]\ \mathit{fix}(\lambda g^n.\, h^n(g^n, -))$$

This follows from the fact that each $\mathcal{R}'[\![\tau'']\!]$ relates $\bot$ to $\bot$ and is closed under least upper bounds of ω-chains, which can be proved by a simple induction on τ''.

The second part follows from Corollary 4.8, by adequacy of the model with respect to the program relation defined in Example 2.3. □

Corollary 4.12. *For our nondeterminism example, we have*

$$\mathcal{V}(\!|e|\!) \ltimes^{\Gamma}_{\mathcal{M}} \mathcal{N}(\!|e|\!)$$

for every $\Gamma \vdash e : \tau$*, and in particular,*

$$\forall V : \mathbf{bool}.\ (\mathcal{V}(\!|e'|\!) \Downarrow \mathbf{return}\, V) \ \Rightarrow\ (\mathcal{N}(\!|e'|\!) \Downarrow \mathbf{return}\, V)$$

for every closed expression $e' : \mathbf{bool}$*.*

Proof. We first extend the inductive proof of Theorem 4.7 with two extra cases: one for **fail** and one for **or**. Following Lemma 4.9 above, to prove both of these cases, it suffices to show that $\mathcal{R}'[\![\tau]\!]$ is closed under finite joins for each τ', i.e. that

$$\bot\, \mathcal{R}'[\![\tau']\!]\, \bot \qquad t_1^v\, \mathcal{R}'[\![\tau']\!]\, t_1^n \ \wedge\ t_2^v\, \mathcal{R}'[\![\tau']\!]\, t_2^n \ \Rightarrow\ (t_1^v \sqcup t_2^v)\, \mathcal{R}'[\![\tau']\!]\, (t_1^n \sqcup t_2^n)$$

This is a simple induction on τ'. The result follows from Theorem 4.7 and Corollary 4.8, by adequacy of the model with respect to the program relation defined in Example 2.4. □

Corollary 4.13. *For our immutable state example, we have*

$$\mathcal{V}(\!|e|\!) \ltimes^{\Gamma}_{\mathcal{M}} \mathcal{N}(\!|e|\!)$$

for every $\Gamma \vdash e : \tau$*, and in particular,*

$$\forall b \in \{\mathit{true}, \mathit{false}\}.\, \exists V : \mathbf{bool}.\ (\mathcal{V}(\!|e|\!) \Downarrow_b \mathbf{return}\, V) \wedge (\mathcal{N}(\!|e|\!) \Downarrow_b \mathbf{return}\, V)$$

for every closed expression $e' : \mathbf{bool}$*.*

Proof. Once again, we need to add the extra case to Theorem 4.7. It is enough to show that $(\mathbf{get} \circ \langle\rangle_{\mathcal{V}[\![\Gamma]\!]} \circ \langle\!\langle f_i^v \rangle\!\rangle_i)\ \mathcal{R}'[\![\mathbf{bool}]\!]\ (\mathbf{get} \circ \langle\rangle_W)$, which follows from lax discardability. We can then apply Theorem 4.7 and Corollary 4.8 to obtain the result, using adequacy with respect to the program relation defined in Example 2.5. □

5. A Galois connection between call-by-value and call-by-name

We improve on the results of the previous section by showing how to directly relate the call-by-value semantics $\mathcal{V}[\![e]\!]$ of an expression to a morphism derived from the call-by-name semantics $\mathcal{N}[\![e]\!]$. Under a further condition on the model (which again restricts the allowable side-effects), we prove a statement (Theorem 5.10) of the form

$$\mathcal{V}[\![e]\!] \ \sqsubseteq\ \psi_\tau \circ \mathcal{N}[\![e]\!] \circ \hat{\phi}_\Gamma$$

involving morphisms ϕ_τ and ψ_τ for mapping between the call-by-value and call-by-name semantics:

$$T(\mathcal{V}[\![\tau]\!]) \underset{\psi_\tau}{\overset{\phi_\tau}{\rightleftarrows}} U_{\mathsf{T}}(\mathcal{N}[\![\tau]\!])$$

(In the inequality above, $\hat{\phi}_\Gamma : \mathcal{V}[\![\Gamma]\!] \to \mathcal{N}[\![\Gamma]\!]$ is constructed by extending the morphisms $(\phi_\tau \circ \eta) : \mathcal{V}[\![\tau]\!] \to U_{\mathsf{T}}(\mathcal{N}[\![\tau]\!])$ from types to contexts.)

We do not want just *any* maps between call-by-value and call-by-name. We show (Corollary 5.9) that the maps we define (precisely, the monotone functions $\phi_\tau \circ (-)$ and $\psi_\tau \circ (-)$) form *Galois connections* [MSS86]. This is the crucial property that enables us to prove the inequality above.[3]

Definition 5.1. A *Galois connection* consists of two posets X, Y and two monotone functions $\phi : X \to Y$, $\psi : Y \to X$, such that $x \sqsubseteq \psi(\phi\, x)$ for all $x \in X$ and $\phi(\psi\, y) \sqsubseteq y$ for all $y \in Y$.

The results of the previous section are helpful here. We have relations $\mathcal{R}[\![\tau]\!]$ that in some sense capture the relationship between call-by-value and call-by-name. This suggests we should look for morphisms ϕ_τ and ψ_τ that represent the relations $\mathcal{R}[\![\tau]\!]$, i.e. that satisfy the following equivalences.

$$\phi_\tau \circ f^v \ \sqsubseteq\ f^n \quad \Leftrightarrow \quad f^v\ \mathcal{R}[\![\tau]\!]_W\ f^n \quad \Leftrightarrow \quad f^v \ \sqsubseteq\ \psi_\tau \circ f^n$$

[3]The proof of Theorem 5.10 that we give here does not directly use the fact that the maps are Galois connections; instead, it uses Theorem 4.7. This is simply to avoid another induction on expressions. In the conference version the corresponding fact [MM22, Lemma 20] was proved directly using the fact that the maps are Galois connections.

$$\boxed{\phi_\tau : T(\mathcal{V}[\![\tau]\!]) \to U_{\mathsf{T}}(\mathcal{N}[\![\tau]\!])}$$

$$\begin{aligned}
\phi_{\mathbf{unit}} &= id_1 \ : \ T1 \to T1 \\
\phi_{\tau_1 \times \tau_2} &= \langle \phi_{\tau_1} \circ T\pi_1, \phi_{\tau_2} \circ T\pi_2 \rangle \ : \ T(\mathcal{V}[\![\tau_1]\!] \times \mathcal{V}[\![\tau_2]\!]) \to U_{\mathsf{T}}(\mathcal{N}[\![\tau_1]\!]) \times U_{\mathsf{T}}(\mathcal{N}[\![\tau_2]\!]) \\
\phi_{\mathbf{bool}} &= id_2 \ : \ T2 \to T2 \\
\phi_{\tau \to \tau'} &= \Lambda(\phi_{\tau'} \circ ev^\dagger \circ \langle\!\langle \pi_1, \psi_\tau \circ \pi_2 \rangle\!\rangle) \ : \ T(\mathcal{V}[\![\tau]\!] \Rightarrow T(\mathcal{V}[\![\tau']\!])) \to U_{\mathsf{T}}(\mathcal{N}[\![\tau]\!]) \Rightarrow U_{\mathsf{T}}(\mathcal{N}[\![\tau']\!])
\end{aligned}$$

$$\boxed{\psi_\tau : U_{\mathsf{T}}(\mathcal{N}[\![\tau]\!]) \to T(\mathcal{V}[\![\tau]\!])}$$

$$\begin{aligned}
\psi_{\mathbf{unit}} &= id_{T1} \ : \ T1 \to T1 \\
\psi_{\tau_1 \times \tau_2} &= \langle\!\langle \psi_{\tau_1} \circ \pi_1, \psi_{\tau_2} \circ \pi_2 \rangle\!\rangle \ : \ U_{\mathsf{T}}(\mathcal{N}[\![\tau_1]\!]) \times U_{\mathsf{T}}(\mathcal{N}[\![\tau_2]\!]) \to T(\mathcal{V}[\![\tau_1]\!] \times \mathcal{V}[\![\tau_2]\!]) \\
\psi_{\mathbf{bool}} &= id_{T2} \ : \ T2 \to T2 \\
\psi_{\tau \to \tau'} &= \eta_{\mathcal{V}[\![\tau \to \tau']\!]} \circ (\Lambda(\psi_{\tau'} \circ ev \circ (id \times (\phi_\tau \circ \eta)))) \\
&\qquad : \ U_{\mathsf{T}}(\mathcal{N}[\![\tau]\!]) \Rightarrow U_{\mathsf{T}}(\mathcal{N}[\![\tau']\!]) \to T(\mathcal{V}[\![\tau]\!] \Rightarrow T(\mathcal{V}[\![\tau']\!]))
\end{aligned}$$

FIGURE 7. Semantic morphisms ϕ from call-by-value to call-by-name and ψ from call-by-name to call-by-value

These equivalences uniquely determine ψ_τ and ϕ_τ, and guarantee that we have Galois connections $(\psi_\tau \circ (-), \phi_\tau \circ (-))$. Furthermore, these equivalences enable us to prove

$$\mathcal{V}[\![e]\!] \ \sqsubseteq \ \psi_\tau \circ \mathcal{N}[\![e]\!] \circ \hat{\phi}_\Gamma$$

as a corollary of $\mathcal{V}(\!|e|\!) \ltimes^\Gamma_{\mathcal{M}} \mathcal{N}(\!|e|\!)$. That is, the main result of this section (Theorem 5.10) is a corollary of the main result of the previous section (Theorem 4.7).

Given a CBPV model $\mathcal{M} = (\mathbf{C}, \mathsf{T})$, the morphisms ϕ_τ and ψ_τ are defined in Figure 7.[4] As for the relations $\mathcal{R}[\![\tau]\!]$, the definition is by induction on τ. This is mutual induction: at contravariant positions, the definition of ϕ uses ψ, and vice-versa.

Of course we do not expect to be able to prove the properties outlined above for a general model $\mathcal{M}$. To see what conditions we should require $\mathcal{M}$ to satisfy, suppose that we do have Galois connections $(\phi_\tau \circ (-), \psi_\tau \circ (-))$, equivalently, that we have

$$\phi_\tau \circ \psi_\tau \sqsubseteq id_{U_{\mathsf{T}}(\mathcal{N}[\![\tau]\!])} \qquad id_{T(\mathcal{V}[\![\tau]\!])} \sqsubseteq \psi_\tau \circ \phi_\tau$$

Now consider what happens when we convert a lazy pair $N = \lambda\{1.\, N_1, 2.\, N_2\}$ of type $\mathcal{N}(\!|\mathbf{unit} \times \mathbf{unit}|\!) = \mathbf{F}\,\mathbf{unit} \mathbin{\underline{\times}} \mathbf{F}\,\mathbf{unit}$ into call-by-value, and then back into call-by-name:

$$\left[\!\!\left[\begin{array}{l} \lambda\{1.\, N_1 \ \mathbf{to}\ z_1.\, N_2 \ \mathbf{to}\ z_2.\, \mathbf{return}\, z_1, \\ \ \ 2.\, N_1 \ \mathbf{to}\ z_1.\, N_2 \ \mathbf{to}\ z_2.\, \mathbf{return}\, z_2\} \end{array} \right]\!\!\right] \ = \ (\phi_{\mathbf{unit}\times\mathbf{unit}} \circ \psi_{\mathbf{unit}\times\mathbf{unit}} \circ [\![N]\!]) \ \sqsubseteq \ [\![N]\!]$$

The ith projection of the left-hand side evaluates both N_1 and N_2, but the ith projection of the right is just $[\![N_i]\!]$. Thus moving from left to right discards side-effects. Similarly, converting a strict pair M of type $\mathbf{F}\,(\mathcal{V}(\!|\mathbf{unit} \times \mathbf{unit}|\!)) = \mathbf{F}\,(\mathbf{unit} \times \mathbf{unit})$ to call-by-name

[4] We present the definitions in a different way to [MM22], but the morphisms are in fact the same. The syntactic maps in Figure 8 below are similarly presented differently to [MM22].

and back duplicates the side-effects of M:

$$[\![M]\!] \sqsubseteq (\psi_{\mathbf{unit}\times\mathbf{unit}} \circ \phi_{\mathbf{unit}\times\mathbf{unit}} \circ [\![M]\!]) = \left[\!\!\left[\begin{array}{l} M \textbf{ to } z \,.\, \textbf{match } z \textbf{ with } (z_1, z_2). \\ M \textbf{ to } z' . \textbf{match } z' \textbf{ with } (z_1', z_2'). \\ \textbf{return } (z_1, z_2') \end{array}\right]\!\!\right]$$

These suggest that lax discardability and lax copyability will be useful, and indeed we use both of these properties in the proof of Lemma 5.8 below.

For function types we need even more. Consider what happens when we convert a CBPV computation $M : \mathbf{F}(\mathcal{V}(\!|\mathbf{unit} \to \mathbf{unit}|\!)) = \mathbf{F}(\mathbf{U}(\mathbf{unit} \to \mathbf{F}\,\mathbf{unit}))$ to call-by-name and then back to call-by-value. By doing this we obtain the denotation of a computation that immediately returns:

$$[\![M]\!] \sqsubseteq (\psi_{\mathbf{unit}\to\mathbf{unit}} \circ \phi_{\mathbf{unit}\to\mathbf{unit}} \circ [\![M]\!]) = [\![\mathbf{return}\,\mathbf{thunk}\,\lambda x\!:\!\mathbf{unit}.\, M \textbf{ to } z.\, x\,\text{`}\,\mathbf{force}\,z]\!]$$

The round-trip from call-by-value to call-by-name and back thunks the side-effects of M, suspending them until the function is applied. The property we ask for the model to satisfy in order to make this a valid inequality is *lax thunkability* of morphisms.

Definition 5.2. Let T be a strong **Poset**-monad on a cartesian **Poset**-category $\mathbf{C}$. A morphism $f : X \to TY$ is *lax thunkable* if $T\eta_Y \circ f \sqsubseteq \eta_{TY} \circ f$. If every such f is lax thunkable (equivalently, if $T\eta_Y \sqsubseteq \eta_{TY}$ for every $Y \in |\mathbf{C}|$), then we say that T is *lax idempotent*.[5]

Again this a lax version of a property defined by Führmann [Füh99]. For the corresponding property in the syntax of CBPV we have the following.

Lemma 5.3. *Let $\Gamma \vdash_c M : \mathbf{F}\,A$ be a computation. For every adequate CBPV model, if $[\![M]\!]$ is lax thunkable, then*

$$M \textbf{ to } x.\, \mathbf{return}\,\mathbf{thunk}\,\mathbf{return}\,x \;\preccurlyeq^{\Gamma}_{\mathrm{ctx}}\; \mathbf{return}\,\mathbf{thunk}\,M$$

Proof. We have

$$\begin{aligned} [\![M \textbf{ to } x.\, \mathbf{return}\,\mathbf{thunk}\,\mathbf{return}\,x]\!] &= T\eta_{[\![A]\!]} \circ [\![M]\!] \\ &\sqsubseteq \eta_{T[\![A]\!]} \circ [\![M]\!] \\ &= [\![\mathbf{return}\,\mathbf{thunk}\,M]\!] \end{aligned}$$

which implies the result by adequacy. □

Example 5.4. For three of our examples the strong **Poset**-monad T is lax idempotent. For no side-effects, we use the identity monad, which is trivially lax idempotent because $T\eta_Y = id_Y = \eta_{TY}$. For divergence, the monad (Example 3.9) is lax idempotent because the left hand side of $T\eta_Y t \sqsubseteq \eta_{TY} t$ is $\bot$ when $t = \bot$ (intuitively, we can thunk diverging computations), and otherwise the two sides are equal. For nondeterminism the monad (Example 3.9) is lax idempotent because, since $\downarrow\{y\} \subseteq S$ for every $y \in S \in TY$, we have

$$T\eta_Y S = \downarrow\{\downarrow\{y\} \mid y \in S\} \subseteq \downarrow\{S\} = \eta_{TY} S$$

(intuitively, we can postpone nondeterministic choices).

On the other hand, the reader monad we use for immutable state (Example 3.11) is *not* lax idempotent. Indeed, a morphism $f : X \to 2 \Rightarrow Y$ is lax thunkable exactly when it satisfies $f\,x\,true = f\,x\,false$ for all $x \in X$. In particular, $[\![\mathbf{get}]\!]$ is not lax thunkable.

[5] Lax idempotent **Poset**-monads are a special case of lax idempotent 2-monads, which are well-known, and are often called *Kock-Zöberlein* monads [Koc95].

As a consequence, we cannot apply the results of this section to this model. (In fact, Proposition 5.11 below implies that the conclusion of Theorem 5.10 is false in this case.) This does not mean that our reasoning principle (Theorem 6.2) does not apply to immutable state, only that this model is not good enough to instantiate it. (It is known that this model of immutable state fails to be fully abstract, i.e. that it distinguishes between computations that are contextually equivalent [KKS22].)

Lax thunkability is difficult to use directly in proofs, so we establish the following characterizations of lax thunkable morphisms.

Lemma 5.5. *Let* T *be a strong* **Poset***-monad and* $f : X \to TY$ *be a morphism. The following are equivalent:*

(1) *f is lax thunkable;*
(2) *the implication*

$$g_1 \sqsubseteq (g_2 \circ (id_W \times \eta_Y))^{\ddagger^{W\times\square}} \quad \Rightarrow \quad g_1 \circ (id_W \times f) \sqsubseteq g_2 \circ (id_W \times f)$$

holds for all T-algebras Z and morphisms $g_1, g_2 : W \times TY \to Z$;
(3) *the implication*

$$g_1 \sqsubseteq (g_2 \circ (\eta_Y \times id_W))^{\ddagger^{\square\times W}} \quad \Rightarrow \quad g_1 \circ (f \times id_W) \sqsubseteq g_2 \circ (f \times id_W)$$

holds for all T-algebras Z and morphisms $g_1, g_2 : TY \times W \to Z$;
(4) *the implication*

$$g_1 \sqsubseteq (g_2 \circ \eta_Y)^{\ddagger} \quad \Rightarrow \quad g_1 \circ f \sqsubseteq g_2 \circ f$$

holds for all T-algebras Z and morphisms $g_1, g_2 : TY \to Z$.

Proof. (1) ⇒ (2): If $g_1 \sqsubseteq (g_2 \circ (id_W \times \eta_Y))^{\ddagger^{W\times\square}}$ then

$$\begin{aligned}
g_1 \circ (id_W \times f) &\sqsubseteq (g_2 \circ (id_W \times \eta_Y))^{\ddagger^{W\times\square}} \circ (id_W \times f) && \text{(assumption)}\\
&= {g_2}^{\ddagger^{W\times\square}} \circ (id_W \times T\eta_Y) \circ (id_W \times f) && \text{(naturality of extension)}\\
&\sqsubseteq {g_2}^{\ddagger^{W\times\square}} \circ (id_W \times \eta_{TY}) \circ (id_W \times f) && \text{(lax thunkability of } f\text{)}\\
&= g_2 \circ (id_W \times f) && \text{(left unit law)}
\end{aligned}$$

(2) ⇒ (4): Specializing (2) to $W = 1$ yields (4).

(4) ⇒ (1): Consider the T-algebra $\mathsf{Z} = F_{\mathsf{T}}(TY)$ and morphisms $g_1 = T\eta_Y$ and $g_2 = \eta_{TY}$. We have $g_1 = T\eta_Y = (\eta_{TY} \circ \eta_Y)^\dagger = (g_2 \circ \eta_Y)^\dagger$ by the definition of T on morphisms, so (3) gives us the required inequality $g_1 \circ f \sqsubseteq g_2 \circ f$.

(1) ⇒ (3): Similar to the proof that (1) implies (2).

(3) ⇒ (4): Similar to the proof that (2) implies (4). □

Lax thunkability is a strong property. In particular, it implies all of the properties we assumed in the previous section. (The non-lax version of this fact is noted by Führmann in [Füh99].)

Lemma 5.6. *Let* T *be a strong* **Poset***-monad. If $f : X \to TY$ is lax thunkable, then f is also lax discardable, lax copyable, and lax central.*

Proof. Lax discardability: By the definition of T on morphisms, we have

$$T\langle\rangle_Y \;=\; (\eta_1 \circ \langle\rangle_Y)^\dagger \;=\; ((\eta_1 \circ \langle\rangle_{TY}) \circ \eta_Y)^\dagger$$

so Lemma 5.5(4) implies

$$T\langle\rangle_Y \circ f \;\sqsubseteq\; (\eta_1 \circ \langle\rangle_{TY}) \circ f \;=\; \eta_1 \circ \langle\rangle_X$$

Lax copyability: By the definition of T on morphisms, and the left unit law for T, we have

$$T\langle id_Y, id_Y\rangle = (\eta_{Y\times Y} \circ \langle id_Y, id_Y\rangle)^\dagger = (seq^{\mathrm{L}}_{Y,Y} \circ \langle \eta_Y, \eta_Y\rangle)^\dagger = ((seq^{\mathrm{L}}_{Y,Y} \circ \langle id_Y, id_Y\rangle) \circ \eta_Y)^\dagger$$

so Lemma 5.5(4) implies

$$T\langle id_Y, id_Y\rangle \circ f \;\sqsubseteq\; (seq^{\mathrm{L}}_{Y,Y} \circ \langle id_Y, id_Y\rangle) \circ f \;=\; seq^{\mathrm{L}}_{Y,Y} \circ \langle f, f\rangle$$

Lax centrality: We have

$$\begin{aligned}
seq^{\mathrm{L}}_{Y,W} &= (\eta_{Y\times W}{}^{\dagger^{Y\times\Box}})^{\dagger^{\Box\times TW}} && \text{(definition of } seq^{\mathrm{L}}\text{)}\\
&= ((\eta_{Y\times W}{}^{\dagger^{\Box\times W}} \circ (\eta_Y \times id_W))^{\dagger^{Y\times\Box}})^{\dagger^{\Box\times TW}} && \text{(left unit law)}\\
&= ((\eta_{Y\times W}{}^{\dagger^{\Box\times W}})^{\dagger^{TY\times\Box}} \circ (\eta_Y \times id_{TW}))^{\dagger^{\Box\times TW}} && \text{(naturality of extension)}\\
&= (seq^{\mathrm{R}}_{Y,W} \circ (\eta_Y \times id_{TW}))^{\dagger^{\Box\times TW}} && \text{(definition of } seq^{\mathrm{R}}\text{)}
\end{aligned}$$

so Lemma 5.5(3) implies

$$seq^{\mathrm{L}}_{Y,W} \circ (f \times id_{TW}) \;\sqsubseteq\; seq^{\mathrm{R}}_{Y,W} \circ (f \times id_{TW}) \qquad \square$$

Our aim is now to establish the relationship between $\mathcal{R}[\![\tau]\!]$ and (ϕ_τ, ψ_τ) outlined at the beginning of this section. For $\tau = \mathbf{unit}$ and $\tau = \mathbf{bool}$ this turns out to be easy. For product types, we use lax discardablity and lax copyability, while for function types, we use lax thunkability. For the latter two cases the following lemma is useful.

Lemma 5.7. *Let T be a strong* **Poset***-monad.*

(1) *If $f : W \to T(X_1 \times X_2)$ is lax copyable, and both $g_1 : W \to TX_1$ and $g_2 : W \to TX_2$ are lax discardable, then*

$$f \sqsubseteq \langle\!\langle g_1, g_2\rangle\!\rangle \quad\Leftrightarrow\quad T\pi_1 \circ f \sqsubseteq g_1 \;\wedge\; T\pi_2 \circ f \sqsubseteq g_2$$

(2) *If $f : W \to T(X \Rightarrow Z)$ is lax thunkable, where Z is a T-algebra, then for every $g : W \times X \to Z$, we have*

$$f \;\sqsubseteq\; \eta_{X\Rightarrow Z} \circ \Lambda g \quad\Leftrightarrow\quad ev^{\ddagger^{\Box\times X}} \circ (f \times id_X) \;\sqsubseteq\; g$$

Proof. (1) For the ($\Rightarrow$) direction, we have

$$\begin{aligned}
T\pi_i \circ f &\sqsubseteq T\pi_i \circ \langle\!\langle g_1, g_2\rangle\!\rangle && \text{(assumption)}\\
&\sqsubseteq g_i && \text{(Lemma 4.6(1))}
\end{aligned}$$

for each $i \in \{1, 2\}$. For the ($\Leftarrow$) direction, we have

$$\begin{aligned}
f &= T(\pi_1 \times \pi_2) \circ T\langle id, id\rangle \circ f\\
&\sqsubseteq T(\pi_1 \times \pi_2) \circ seq^{\mathrm{L}} \circ \langle f, f\rangle && \text{(lax copyability)}\\
&= \langle\!\langle T\pi_1 \circ f, T\pi_2 \circ f\rangle\!\rangle && \text{(naturality of extension)}\\
&\sqsubseteq \langle\!\langle g_1, g_2\rangle\!\rangle && \text{(assumption)}
\end{aligned}$$

(2) For the ($\Rightarrow$) direction, we have

$$\begin{aligned} ev^{\ddagger^{\square\times X}} \circ (f \times id_X) &\sqsubseteq ev^{\ddagger^{\square\times X}} \circ ((\eta_{X\Rightarrow Z} \circ \Lambda g) \times id_X) && \text{(assumption)} \\ &= ev \circ (\Lambda g \times id_X) && \text{(left unit law)} \\ &= g \end{aligned}$$

For the ($\Leftarrow$) direction, we note that

$$\begin{aligned} \eta_{X\Rightarrow Z} \circ \Lambda(ev^{\ddagger^{\square\times X}}) \circ f &= \eta_{X\Rightarrow Z} \circ \Lambda(ev^{\ddagger^{\square\times X}} \circ (f \times id_X)) \\ &\sqsubseteq \eta_{X\Rightarrow Z} \circ \Lambda g && \text{(assumption)} \end{aligned}$$

so it suffices to show $f \sqsubseteq \eta_{X\Rightarrow Z} \circ \Lambda(ev^{\ddagger^{\square\times X}}) \circ f$. Since f is lax thunkable, this is a consequence of Lemma 5.5(4) applied to the following.

$$\begin{aligned} id_{T(X\Rightarrow Z)} &= (\eta_{X\Rightarrow Z} \circ \Lambda ev)^\dagger && \text{(right unit law)} \\ &= (\eta_{X\Rightarrow Z} \circ \Lambda(ev^{\ddagger^{\square\times X}} \circ (\eta_{X\Rightarrow Z} \times id_X)))^\dagger && \text{(left unit law)} \\ &= (\eta_{X\Rightarrow Z} \circ \Lambda(ev^{\ddagger^{\square\times X}}) \circ \eta_{X\Rightarrow Z})^\dagger \end{aligned}$$

□

We can now relate $\mathcal{R}[\![\tau]\!]$ to the morphisms ϕ_τ and ψ_τ, as follows.

Lemma 5.8. *Let $\mathcal{M} = (\mathbf{C}, \mathsf{T})$ be a CBPV model for which T is lax idempotent. For every type τ, object $W \in |\mathbf{C}|$, and pair of morphisms*

$$f^v : W \to T(\mathcal{V}[\![\tau]\!]) \qquad f^n : W \to U_{\mathsf{T}}(\mathcal{N}[\![\tau]\!])$$

we have the following equivalences.

$$\phi_\tau \circ f^v \sqsubseteq f^n \quad \Leftrightarrow \quad f^v \; \mathcal{R}[\![\tau]\!]_W \; f^n \quad \Leftrightarrow \quad f^v \sqsubseteq \psi_\tau \circ f^n$$

Proof. By induction on the type τ.

- The **unit** case is trivial.
- For a product type $\tau_1 \times \tau_2$, we have that $f^v \; \mathcal{R}[\![\tau_1 \times \tau_2]\!] \; f^n$ is equivalent, by expanding out the definition and applying the inductive hypothesis, to each of the following properties.

  $$\forall i \in \{1,2\}.\ \phi_{\tau_i} \circ T\pi_i \circ f^v \sqsubseteq \pi_i \circ f^n \qquad \forall i \in \{1,2\}.\ T\pi_i \circ f^v \sqsubseteq \psi_{\tau_i} \circ \pi_i \circ f^n$$

  It therefore suffices to show that the left is equivalent to $\phi_{\tau_1\times\tau_2} \circ f^v \sqsubseteq f^n$, and that the right is equivalent to $f^v \sqsubseteq \psi_{\tau_1\times\tau_2} \circ f^n$. For the left, the inequality $\phi_{\tau_1\times\tau_2} \circ f^v \sqsubseteq f^n$ holds exactly when $\pi_i \circ \phi_{\tau_1\times\tau_2} \circ f^v \sqsubseteq \pi_i \circ f^n$ holds for all $i \in \{1,2\}$. The required equivalence therefore follows from

  $$\pi_i \circ \phi_{\tau_1\times\tau_2} \circ f^v = T\pi_i \circ \phi_{\tau_i} \circ f^v$$

  which is immediate from the definition of $\phi_{\tau_1\times\tau_2}$. For the right, it is enough to apply Lemma 5.7(1) to the inequality $f^v \sqsubseteq \psi_{\tau_1\times\tau_2} \circ f^n$. We can do this because T is lax idempotent, which implies lax discardability and lax copyability by Lemma 5.6.
- The **bool** case is trivial.
- For a function type $\tau \to \tau'$, we consider the two equivalences separately.

  For the equivalence on the left, we have

  $$\begin{aligned} \phi_{\tau\to\tau'} \circ f^v \sqsubseteq f^n &\Leftrightarrow \Lambda^{-1}(\phi_{\tau\to\tau'} \circ f^v) \sqsubseteq \Lambda^{-1} f^n \\ &\Leftrightarrow \phi_{\tau'} \circ ev^\dagger \circ \langle\!\langle f^v \circ \pi_1, \psi_\tau \circ \pi_2 \rangle\!\rangle \sqsubseteq ev \circ (f^n \times id) \\ &\Leftrightarrow (ev^\dagger \circ \langle\!\langle f^v \circ \pi_1, \psi_\tau \circ \pi_2 \rangle\!\rangle) \; \mathcal{R}[\![\tau']\!]_{W\times U_{\mathsf{T}}(\mathcal{N}[\![\tau]\!])} \; (ev \circ (f^n \times id)) \end{aligned}$$

where the second step uses the definition of $\phi_{\tau\to\tau'}$, and the final step uses the inductive hypothesis. Call the instance of $\mathcal{R}[\![\tau']\!]_{W\times U_{\mathsf{T}}(\mathcal{N}[\![\tau]\!])}$ above $(*)$. To show that $f^v\ \mathcal{R}[\![\tau\to\tau']\!]_W\ f^n$ follows from $(*)$, consider arbitrary $w : W' \to W$ and g^v, g^n such that $g^v\ \mathcal{R}[\![\tau]\!]_{W'}\ g^n$. By Lemma 4.5(2), we can precompose both sides of $(*)$ with $\langle w, g^n\rangle$ to obtain

$$(ev^\dagger \circ \langle\!\langle f^v \circ w, \psi_\tau \circ g^n\rangle\!\rangle)\ \mathcal{R}[\![\tau']\!]_{W'}\ (ev \circ \langle f^n \circ w, g^n\rangle)$$

The inductive hypothesis implies $g^v \sqsubseteq \psi_\tau \circ g^n$, so by Lemma 4.5(1) it follows that

$$(ev^\dagger \circ \langle\!\langle f^v \circ w, g^v\rangle\!\rangle)\ \mathcal{R}[\![\tau']\!]_{W'}\ (ev \circ \langle f^n \circ w, g^n\rangle)$$

as required. Conversely, $f^v\ \mathcal{R}[\![\tau\to\tau']\!]_W\ f^n$ implies $(*)$ by taking

$$W' = W \times U_{\mathsf{T}}(\mathcal{N}[\![\tau]\!]) \qquad w = \pi_1 : W' \to W \qquad g^v = \psi_\tau \circ \pi_2 \qquad g^n = \pi_2$$

and noting that $\psi_\tau \circ \pi_2 \sqsubseteq \psi_\tau \circ \pi_2$ implies $(\psi_\tau \circ \pi_2)\ \mathcal{R}[\![\tau]\!]_{W'}\ \pi_2$ by the inductive hypothesis.

For the remaining equivalence, we have

$$\begin{aligned} f^v \ \sqsubseteq\ \psi_{\tau\to\tau'} \circ f^n \ &\Leftrightarrow\ ev^{\dagger^{\square\times\mathcal{V}[\![\tau]\!]}} \circ (f^v \times id) \ \sqsubseteq\ \psi_{\tau'} \circ ev \circ (f^n \times (\phi_\tau \circ \eta)) \\ &\Leftrightarrow\ (ev^{\dagger^{\square\times\mathcal{V}[\![\tau]\!]}} \circ (f^v \times id))\ \mathcal{R}[\![\tau']\!]_{W\times\mathcal{V}[\![\tau]\!]}\ (ev \circ (f^n \times (\phi_\tau \circ \eta))) \end{aligned}$$

where the first step uses Lemma 5.7(2) and the second uses the inductive hypothesis. Call the instance of $\mathcal{R}[\![\tau']\!]_{W\times\mathcal{V}[\![\tau]\!]}$ above $(**)$. To show that $(**)$ implies $f^v\ \mathcal{R}[\![\tau\to\tau']\!]_W\ f^n$, consider arbitrary $w : W' \to W$ and g^v, g^n such that $g^v\ \mathcal{R}[\![\tau]\!]_{W'}\ g^n$. By Lemma 4.5(2,3), we have

$$(ev^{\dagger^{\square\times\mathcal{V}[\![\tau]\!]}} \circ (f^v \times id))^{\dagger^{W\times\square}} \circ \langle w, g^v\rangle\ \mathcal{R}[\![\tau']\!]_{W'}\ (ev \circ (f^n \times (\phi_\tau \circ \eta)))^{\ddagger^{W\times\square}} \circ \langle w, g^v\rangle$$

Lax idempotence of T implies lax centrality by Lemma 5.6, so we have

$$\begin{aligned} &(ev^{\dagger^{\square\times\mathcal{V}[\![\tau]\!]}} \circ (f^v \times id))^{\dagger^{W\times\square}} \circ \langle w, g^v\rangle \\ =\ &(ev^{\dagger^{\square\times\mathcal{V}[\![\tau]\!]}})^{\dagger^{T(\mathcal{V}[\![\tau\to\tau']\!])\times\square}} \circ \langle f^v \circ w, g^v\rangle && \text{(naturality of extension)} \\ =\ &(ev^{\dagger^{\mathcal{V}[\![\tau\to\tau']\!]\times\square}})^{\dagger^{\square\times T(\mathcal{V}[\![\tau]\!])}} \circ \langle f^v \circ w, g^v\rangle && \text{(Lemma 4.6(2))} \\ =\ &ev^\dagger \circ \langle\!\langle f^v \circ w, g^v\rangle\!\rangle && \text{(left unit, associativity)} \end{aligned}$$

Since $g^v\ \mathcal{R}[\![\tau]\!]_{W'}\ g^n$, we also have

$$\begin{aligned} (ev \circ (f^n \times (\phi_\tau \circ \eta)))^{\ddagger^{W\times\square}} \circ \langle w, g^v\rangle \ &\sqsubseteq\ (ev \circ (f^n \times \phi_\tau)) \circ \langle w, g^v\rangle && \text{(Lemma 5.5(2))} \\ &=\ ev \circ \langle f^n \circ w, \phi_\tau \circ g^v\rangle \\ &\sqsubseteq\ ev \circ \langle f^n \circ w, g^n\rangle && \text{(inductive hypothesis)} \end{aligned}$$

where we again use the fact that T is lax idempotent. It follows by Lemma 4.5(1) that

$$(ev^\dagger \circ \langle\!\langle f^v \circ w, g^v\rangle\!\rangle)\ \mathcal{R}[\![\tau']\!]_{W'}\ (ev \circ \langle f^n \circ w, g^n\rangle)$$

as required. Finally, to show that $f^v\ \mathcal{R}[\![\tau\to\tau']\!]_W\ f^n$ implies $(**)$, we take

$$W' = W \times \mathcal{V}[\![\tau]\!] \qquad w = \pi_1 \qquad g^v = \eta \circ \pi_2 \qquad g^n = \phi_\tau \circ \eta \circ \pi_2$$

noting that the inductive hypothesis implies $(\eta \circ \pi_2)\ \mathcal{R}[\![\tau]\!]_{W'}\ (\phi_\tau \circ \eta \circ \pi_2)$. From this we obtain

$$(ev^\dagger \circ \langle\!\langle f^v \circ \pi_1, \eta \circ \pi_2\rangle\!\rangle)\ \mathcal{R}[\![\tau']\!]_{W\times\mathcal{V}[\![\tau]\!]}\ (ev \circ (f^n \times (\phi_\tau \circ \eta)))$$

which simplifies to $(**)$ by the left unit law. □

An immediate corollary is that, as claimed above, the maps between call-by-value and call-by-name form Galois connections.

Corollary 5.9. *Let $\mathcal{M} = (\mathbf{C}, \mathsf{T})$ be a CBPV model such that T is lax idempotent. The monotone functions*

$$\mathbf{C}(W, T(\mathcal{V}[\![\tau]\!])) \underset{\psi_\tau \circ (-)}{\overset{\phi_\tau \circ (-)}{\rightleftarrows}} \mathbf{C}(W, U_{\mathsf{T}}(\mathcal{N}[\![\tau]\!]))$$

form a Galois connection for every source-language type τ and object $W \in |\mathbf{C}|$.

Proof. Immediate from Lemma 5.8. □

Another corollary of Lemma 5.8 is the following, which is the main result of this section. We use this result in the following section to establish our reasoning principle. To state it, we use morphisms $\hat{\phi}_\Gamma$ for converting a call-by-value context into a call-by-name context, defined by

$$\hat{\phi}_\Gamma = \langle \phi_{\tau_i} \circ \eta_{\mathcal{V}[\![\tau_i]\!]} \rangle_i : \mathcal{V}[\![\Gamma]\!] \to \mathcal{N}[\![\Gamma]\!]$$

where $\Gamma = x_1 : \tau_1, \ldots, x_k : \tau_k$.

Theorem 5.10. *Let $\mathcal{M} = (\mathbf{C}, \mathsf{T})$ be a CBPV model such that T is lax idempotent. For all source-language expressions $\Gamma \vdash e : \tau$ we have*

$$\mathcal{V}[\![e]\!] \;\sqsubseteq\; \psi_\tau \circ \mathcal{N}[\![e]\!] \circ \hat{\phi}_\Gamma$$

Proof. The inequality we need to establish is equivalently

$$\mathcal{V}[\![e]\!] \sqsubseteq \psi_\tau \circ \mathcal{N}[\![e]\!] \circ \langle \phi_{\tau_i} \circ \eta_{\mathcal{V}[\![\tau_i]\!]} \circ \pi_{x_i} \rangle_i$$

where $\Gamma = x_1 : \tau_1, \ldots, x_k : \tau_k$. By Lemma 5.8 we have

$$(\eta_{\mathcal{V}[\![\tau_i]\!]} \circ \pi_{x_i})\ \mathcal{R}[\![\tau_i]\!]_{\mathcal{V}[\![\Gamma]\!]}\ (\phi_{\tau_i} \circ \eta_{\mathcal{V}[\![\tau_i]\!]} \circ \pi_{x_i})$$

for each i. We invoke Theorem 4.7 (using Lemma 5.6 to show lax discardability, lax copyability and lax centrality), to obtain

$$(\mathcal{V}[\![e]\!]^\dagger \circ \langle\!\langle \eta_{\mathcal{V}[\![\tau_i]\!]} \circ \pi_{x_i} \rangle\!\rangle_i)\ \mathcal{R}[\![\tau]\!]_{\mathcal{V}[\![\Gamma]\!]}\ (\mathcal{N}[\![e]\!] \circ \langle \phi_{\tau_i} \circ \eta_{\mathcal{V}[\![\tau_i]\!]} \circ \pi_{x_i} \rangle_i)$$

The left-hand side is equal to $\mathcal{V}[\![e]\!]$ by the left unit law of T, so Lemma 5.8 implies the required inequality. □

This theorem has a partial converse, as follows.

Proposition 5.11. *Let $\mathcal{M} = (\mathbf{C}, \mathsf{T})$ be an arbitrary CBPV model. If $\mathcal{V}[\![e]\!] \sqsubseteq \psi_\tau \circ \mathcal{N}[\![e]\!] \circ \hat{\phi}_\Gamma$ for every $\Gamma \vdash e : \tau$, then for each object $X \in \{1, 2\}$ we have $T\eta_X \sqsubseteq \eta_{TX}$, and every morphism $Y \to TX$ is lax thunkable.*

Proof. The first step is to show that $id \sqsubseteq \psi_\tau \circ \phi_\tau$ for every τ, by applying the assumption to the expression $x : \mathbf{unit} \to \tau \vdash x\,() : \tau$. Indeed, we have

$$ev \;=\; \mathcal{V}[\![x\,()]\!] \;\sqsubseteq\; \psi_\tau \circ \mathcal{N}[\![x\,()]\!] \circ \hat{\phi}_{\mathbf{unit}\to\tau} \;=\; \psi_\tau \circ \phi_\tau \circ ev$$

which implies $id \sqsubseteq \psi_\tau \circ \phi_\tau$ because $ev : 1 \times (1 \Rightarrow T(\mathcal{V}[\![\tau]\!])) \to T(\mathcal{V}[\![\tau]\!])$ is an isomorphism. It follows for each $\tau' \in \{\mathbf{unit}, \mathbf{bool}\}$ that

$$id_{T(1 \Rightarrow T(\mathcal{V}[\![\tau']\!]))} \sqsubseteq \psi_{\mathbf{unit}\to\tau'} \circ \phi_{\mathbf{unit}\to\tau'} = \eta_{1 \Rightarrow T(\mathcal{V}[\![\tau']\!])} \circ id_{1 \Rightarrow T(\mathcal{V}[\![\tau']\!])}{}^{\ddagger}$$

$$\boxed{\Gamma \vdash_c M : \mathbf{F}\,(\mathcal{V}(\!|\tau|\!)) \;\mapsto\; \Gamma \vdash_c \Phi_\tau M : \mathcal{N}(\!|\tau|\!)}$$

$$\begin{aligned}
\Phi_{\mathbf{unit}} M &= M \\
\Phi_{\tau_1 \times \tau_2} M &= \lambda\{1.\, M \;\mathbf{to}\; x.\, \mathbf{match}\; x \;\mathbf{with}\; (z_1, z_2).\, (\hat{\Phi}_{\tau_1} z_1), \\
&\qquad 2.\, M \;\mathbf{to}\; x.\, \mathbf{match}\; x \;\mathbf{with}\; (z_1, z_2).\, (\hat{\Phi}_{\tau_2} z_2)\} \\
\Phi_{\mathbf{bool}} M &= M \\
\Phi_{\tau \to \tau'} M &= \lambda x{:}\mathbf{U}\,(\mathcal{N}(\!|\tau|\!)).\, M \;\mathbf{to}\; f.\; \Psi_\tau(\mathbf{force}\, x) \;\mathbf{to}\; y.\; \Phi_{\tau'}(y \,`\, \mathbf{force}\, f)
\end{aligned}$$

$$\boxed{\Gamma \vdash_c N : \mathcal{N}(\!|\tau|\!) \;\mapsto\; \Gamma \vdash_c \Psi_\tau N : \mathbf{F}\,(\mathcal{V}(\!|\tau|\!))}$$

$$\begin{aligned}
\Psi_{\mathbf{unit}} N &= N \\
\Psi_{\tau_1 \times \tau_2} N &= \Psi_{\tau_1}(1\,`N) \;\mathbf{to}\; z_1.\, \Psi_{\tau_2}(2\,`N) \;\mathbf{to}\; z_2.\, \mathbf{return}\,(z_1, z_2) \\
\Psi_{\mathbf{bool}} N &= N \\
\Psi_{\tau \to \tau'} N &= \mathbf{return}\,\mathbf{thunk}\,\lambda x{:}\mathcal{V}(\!|\tau|\!).\, \Psi_{\tau'}\big((\mathbf{thunk}\,(\Phi_\tau(\mathbf{return}\, x)))\,`\, N\big)
\end{aligned}$$

FIGURE 8. Syntactic maps Φ from call-by-value to call-by-name and Ψ from call-by-name to call-by-value

which implies

$$id_{T(T(\mathcal{V}[\![\tau']\!]))} \sqsubseteq \eta_{T(\mathcal{V}[\![\tau']\!])} \circ id_{T(\mathcal{V}[\![\tau']\!])}{}^{\dagger}$$

Hence, by naturality of Kleisli extension and the right unit law, we have

$$T\eta_{\mathcal{V}[\![\tau']\!]} \;=\; id_{T(T(\mathcal{V}[\![\tau']\!]))} \circ T\eta_{\mathcal{V}[\![\tau']\!]} \;\sqsubseteq\; \eta_{T(\mathcal{V}[\![\tau']\!])} \circ id_{T(\mathcal{V}[\![\tau']\!])}{}^{\dagger} \circ T\eta_{\mathcal{V}[\![\tau']\!]} \;=\; \eta_{T(\mathcal{V}[\![\tau']\!])}$$

as required. □

In particular, it follows from this proposition that lax discardability, lax copyability, and lax centrality are not enough. Our immutable state example satisfies all three of those properties, but the morphism $[\![\mathbf{get}]\!] : 1 \to T2$ is not lax thunkable, so we do not have $\mathcal{V}[\![e]\!] \sqsubseteq \psi_\tau \circ \mathcal{N}[\![e]\!] \circ \hat{\phi}_\Gamma$ for every e.

6. The reasoning principle

We now use the Galois connections defined in the previous section to relate the call-by-value and call-by-name translations of expressions, and arrive at our main reasoning principle.

Recall that the problem with comparing $\mathcal{V}(\!|e|\!)$ with $\mathcal{N}(\!|e|\!)$ directly is that they have different types. We render the Galois connections defined in the previous section in the syntax of CBPV, and then construct from $\mathcal{N}(\!|e|\!)$ a computation that we can directly compare with $\mathcal{V}(\!|e|\!)$:

$$\mathcal{V}(\!|e|\!) \preccurlyeq_{\mathrm{ctx}} \Psi_\tau\big(\mathcal{N}(\!|e|\!)[\hat{\Phi}_\Gamma]\big)$$

More precisely, we render ϕ_τ and ψ_τ in the syntax as maps Φ_τ from call-by-value computations to call-by-name computations, and Ψ_τ from call-by-name to call-by-value.[6] These are defined, again by induction on τ, in Figure 8. (We use some auxiliary variables in

the definition, which are assumed to be fresh.) We further define, for each source-language context $\Gamma = x_1 : \tau_1, \ldots, x_k : \tau_k$, a substitution

$$\hat{\Phi}_\Gamma \;=\; x_1 \mapsto \mathbf{thunk}\left(\Phi_{\tau_1}(\mathbf{return}\, x_1)\right), \ldots, x_k \mapsto \mathbf{thunk}\left(\Phi_{\tau_k}(\mathbf{return}\, x_k)\right)$$

for converting a call-by-value context into a call-by-name context. This has the following typing:

$$\mathcal{N}(\!|\Gamma|\!) \vdash_c N : \underline{C} \;\mapsto\; \mathcal{V}(\!|\Gamma|\!) \vdash_c \hat{\Phi}_\Gamma N : \underline{C}$$

The maps Φ, Ψ and $\hat{\Phi}$ are syntactic renderings of ϕ, ψ and $\hat{\phi}$ in the following sense.

Lemma 6.1. *Given any model of CBPV, we have:*
(1) $[\![\Phi_\tau M]\!] = \phi_\tau \circ [\![M]\!]$ *for all* $\Gamma \vdash_c M : \mathbf{F}\,(\mathcal{V}(\!|\tau|\!))$*;*
(2) $[\![\Psi_\tau N]\!] = \psi_\tau \circ [\![N]\!]$ *for all* $\Gamma \vdash_c N : \mathcal{N}(\!|\tau|\!)$*;*
(3) $[\![N[\hat{\Phi}_\Gamma]]\!] = [\![N]\!] \circ \hat{\phi}_\Gamma$ *for all* $\mathcal{N}(\!|\Gamma|\!) \vdash_c N : \underline{C}$*.*

Proof. (1) and (2) are proved by mutual induction on the type τ, with each case being an easy calculation. (3) then follows immediately from (1) together with the evident substitution lemma for the denotational semantics of CBPV. □

Given a source-language expression $\Gamma \vdash e : \tau$, the computation we obtain by composing $\mathcal{N}(\!|\tau|\!)$ with the maps between call-by-value and call-by-name has the same type as $\mathcal{V}(\!|e|\!)$:

$$\mathcal{V}(\!|\Gamma|\!) \vdash_c \Psi_\tau\big(\mathcal{N}(\!|e|\!)[\hat{\Phi}_\Gamma]\big) \;: \mathbf{F}\,(\mathcal{V}(\!|\tau|\!))$$

We can therefore compare $\mathcal{V}(\!|e|\!)$ with $\Psi_\tau(\mathcal{N}(\!|e|\!)[\hat{\Phi}_\Gamma])$ directly. In particular, it makes sense to replace $\mathcal{V}(\!|e|\!)$ with $\Psi_\tau(\mathcal{N}(\!|e|\!)[\hat{\Phi}_\Gamma])$ within a CBPV computation, as outlined in the introduction. Using the results of the previous section, we establish the following result for reasoning about how replacing $\mathcal{V}(\!|e|\!)$ in this way changes the behaviour of a computation.

Recall that a program relation $\preccurlyeq$ is a preorder on CBPV programs, and that each program relation induces a contextual preorder $\preccurlyeq_{\mathrm{ctx}}$. Given any program relation $\preccurlyeq$, to show that the call-by-value and call-by-name translations of source-language expressions are related by $\preccurlyeq_{\mathrm{ctx}}$ it is enough to find an adequate model involving a lax idempotent T:

Theorem 6.2 (Relationship between call-by-value and call-by-name)**.** *Let* $\mathcal{M} = (\mathbf{C}, \mathsf{T})$ *be a CBPV model that is adequate with respect to a given a program relation* $\preccurlyeq$*. If* T *is lax idempotent, then for every source-language expression* $\Gamma \vdash e : \tau$ *we have*

$$\mathcal{V}(\!|e|\!) \;\preccurlyeq_{\mathrm{ctx}}\; \Psi_\tau\big(\mathcal{N}(\!|e|\!)[\hat{\Phi}_\Gamma]\big)$$

Proof. By adequacy it suffices to show $[\![\mathcal{V}(\!|e|\!)]\!] \sqsubseteq [\![\Psi_\tau(\mathcal{N}(\!|e|\!)[\hat{\Phi}_\Gamma])]\!]$, which, by Lemma 6.1, is equivalently $\mathcal{V}[\![e]\!] \sqsubseteq \psi_\tau \circ \mathcal{N}[\![e]\!] \circ \hat{\phi}_\Gamma$. The result therefore follows from Theorem 5.10. □

The generality of this theorem comes from two sources. First, we consider arbitrary program relations $\preccurlyeq$. The only requirement on these is the existence of some adequate model in which morphisms are lax thunkable. Second, this theorem applies to terms that are open and have higher types, using the maps between the two evaluation orders (in contrast to Corollary 4.8 above).

[6]We define Φ and Ψ directly as maps from computations to computations, but we could instead have defined computations

$$x : \mathbf{U}\,(\mathbf{F}\,(\mathcal{V}(\!|\tau|\!))) \vdash_c \Phi'_\tau : \mathcal{N}(\!|\tau|\!) \qquad x : \mathbf{U}\,(\mathcal{N}(\!|\tau|\!)) \vdash_c \Psi'_\tau : \mathbf{F}\,(\mathcal{V}(\!|\tau|\!))$$

and then recovered Φ and Ψ modulo $\beta\eta$-laws for thunks, by substitution. This definition is slightly less convenient to work with however.

For our first three examples (no side-effects, divergence, and nondeterminism), the model is adequate and has a lax idempotent T. Thus in each case the assumptions of our reasoning principle are satisfied, establishing the claims stated informally at the beginning of the introduction.

Remark 6.3. Given an adequate model in which T is lax idempotent, it follows from Corollary 5.9 and Lemma 6.1 that the maps Φ_τ and Ψ_τ on terms form a Galois connection (with respect to $\preccurlyeq_{\mathrm{ctx}}$). In particular, we have

$$M \preccurlyeq_{\mathrm{ctx}} \Psi_\tau(\Phi_\tau M) \qquad \Phi_\tau(\Psi_\tau N) \preccurlyeq_{\mathrm{ctx}} N$$

Both of these inequalities are in general proper (they are not contextual equivalences). To see this, consider our divergence example, for which the above inequalities hold. For each $\underline{C}$, let $\Omega_{\underline{C}}$ be the diverging computation $\mathbf{rec}\, x : \mathbf{U}\underline{C}.\, \mathbf{force}\, x$ (which has type $\underline{C}$). Then if $\tau = \mathbf{bool} \to \mathbf{bool}$ and $M = \Omega_{\mathbf{F}(\mathcal{V}(\!|\tau|\!))}$, we do not have $M \succcurlyeq_{\mathrm{ctx}} \Psi_\tau(\Phi_\tau M)$, because for $\mathcal{E} = (\Box\ \mathbf{to}\ f.\, \mathbf{return}\, \mathbf{false})$ the computation $\mathcal{E}[M]$ diverges but $\mathcal{E}[\Psi_\tau(\Phi_\tau M)] \Downarrow \mathbf{return}\, \mathbf{false}$. In this case we have $\Psi_\tau(\Phi_\tau M) \cong_{\mathrm{ctx}} \mathbf{return}\, \mathbf{thunk}\, \lambda x : \mathbf{bool}.\, \Omega_{\mathbf{F}\, \mathbf{bool}}$. For a counterexample to $\Phi_\tau(\Psi_\tau N) \succcurlyeq_{\mathrm{ctx}} N$, let $\tau = \mathbf{bool} \to \mathbf{bool}$ and $N = \lambda x : \mathbf{U}(\mathbf{F}\, \mathbf{bool}).\, \mathbf{return}\, \mathbf{true}$. Then for $\mathcal{E}' = ((\mathbf{thunk}\, \Omega_{\mathbf{F}\, \mathbf{bool}})\, `\, \Box)$, the computation $\mathcal{E}'[\Phi_\tau(\Psi_\tau N)]$ diverges but $\mathcal{E}'[N] \Downarrow \mathbf{return}\, \mathbf{true}$. Here we have $\Phi_\tau(\Psi_\tau N) \cong_{\mathrm{ctx}} \lambda x : \mathbf{U}(\mathbf{F}\, \mathbf{bool}).\, \mathbf{force}\, x\ \mathbf{to}\ y.\, \mathbf{return}\, \mathbf{true}$.

In particular, our maps between call-by-value and call-by-name are merely Galois connections, and not sections or retractions. This contrasts with Reynolds [Rey74], who obtains a retraction between direct and continuation semantics.

7. Related work

Comparing evaluation orders. Plotkin [Plo75] and many others (e.g. [IT16]) relate call-by-value and call-by-name. Crucially, they consider lambda-calculi with no side-effects other than divergence. This makes a significant difference to the techniques that can be used, in particular because in this case the equational theory for call-by-name is strictly weaker than for call-by-value. This is not necessarily true for other side-effects. Other evaluation orders (such as call-by-need) have also been compared in similarly restricted settings [MOTW95, MM19, HH19]. We suspect our technique could also be adapted to these. Here we use CBPV as a calculus in which to reason about both call-by-value and call-by-name, but other calculi (e.g. the modal calculus of [SPU22]) may be suitable for this purpose.

It might also be possible to recast some of our work in terms of the *duality* between call-by-value and call-by-name [Fil89, CH00, Wad03, Sel01], In particular, this may shed some light on our definitions of Φ and Ψ. It is not clear to us what the precise connection is however. While Selinger [Sel01] defines translations between call-by-value and call-by-name versions of Parigot's $\lambda\mu$-calculus [Par92], these translations behave differently to ours, in particular, they are semantics-preserving.

Relating semantics of languages. The technique we use here to relate call-by-value and call-by-name is based on the idea used first by Reynolds [Rey74] to relate direct and continuation semantics of the lambda calculus, and later used by others (e.g. [MW85, Kuč98, CF94, Fil96]). Reynolds constructs a relation between the two semantics, and uses this to establish a retraction between direct and continuation semantics, just as we construct a relation between call-by-value and call-by-name and then use this to establish a Galois connection. A minor difference is that Reynolds relies on continuations with a large-enough domain of answers (e.g. a solution to a particular recursive domain equation). Our maps exist for *any* choice of model. We are the first to use this technique to relate call-by-value and call-by-name. There has been some work [SF92, LD93, SW96] on soundness and completeness properties of translations (similar to the translations into CBPV), in particular using Galois connections (and similar structures) for which the order is reduction of programs. Our results would fail if we used reduction of programs directly, so we consider only the observable behaviour of programs.

There are some similarities between our work and the work of New et al. [NL20, NLA21] on gradual typing. In particular, [NLA21] has embedding-projection pairs (a special case of Galois connections) for casting from a more dynamic type to a less dynamic type, and vice versa. Their application is quite different however. The double category perspective used in [NL20] may also be illuminating here.

8. Conclusions

In this paper, we give a general reasoning principle (Theorem 6.2) that relates the observable behaviour of terms under call-by-value and call-by-name. The reasoning principle works for various collections of side-effects, in particular, it enables us to obtain theorems about divergence and nondeterminism. It is about open expressions, and enables us to change evaluation order *within* programs.

The technique we use involves first relating the observable behaviour of the call-by-value and call-by-name translations of expressions via a logical relation (Theorem 4.7). We obtain a result about call-by-value and call-by-name evaluations of *programs* as a corollary (Corollary 4.8). Applying this to divergence, we show that if the call-by-value execution terminates with some result then the call-by-name execution terminates with the same result. For nondeterminism, we show that all possible results of call-by-value executions are possible results of call-by-name executions. There may be other collections of side-effects we can apply our technique to, including combinations of divergence and nondeterminism.

We expect that our technique can be applied to other evaluation orders. Two evaluation orders can be related by giving translations into some common language (here we use CBPV), constructing maps between the two translations, and showing that (for some models) these maps form Galois connections. A major advantage of the technique is that it allows us to identify axiomatic properties of side-effects (thunkable, etc.) that give rise to relationships between evaluation orders.

Acknowledgments

We thank the anonymous referees for helpful comments. The first author was supported by an EPSRC studentship, and by Icelandic Research Fund grants 196323-053 and 228684-052.

References

[CF94] Robert Cartwright and Matthias Felleisen. Extensible denotational language specifications. In *Proceedings of the International Conference on Theoretical Aspects of Computer Software*, pages 244–272. Springer, 1994. doi:10.1007/3-540-57887-0_99.

[CH00] Pierre-Louis Curien and Hugo Herbelin. The duality of computation. In *Proceedings of the Fifth ACM SIGPLAN International Conference on Functional Programming*, pages 233–243. ACM, 2000. doi:10.1145/351240.351262.

[DCL18] Marco Devesas Campos and Paul Blain Levy. A syntactic view of computational adequacy. In Christel Baier and Ugo Dal Lago, editors, *Foundations of Software Science and Computation Structures*, pages 71–87. Springer, 2018. doi:10.1007/978-3-319-89366-2_4.

[Fil89] Andrzej Filinski. Declarative continuations and categorical duality. Master's thesis, University of Copenhagen, 1989.

[Fil96] Andrzej Filinski. *Controlling effects*. PhD thesis, Carnegie Mellon University, 1996.

[Füh99] Carsten Führmann. Direct models of the computational lambda-calculus. *Electronic Notes in Theoretical Computer Science*, 20:245–292, 1999. doi:10.1016/S1571-0661(04)80078-1.

[HH19] Jennifer Hackett and Graham Hutton. Call-by-need is clairvoyant call-by-value. *Proc. ACM Program. Lang.*, 3(ICFP):114:1–114:23, 2019. doi:10.1145/3341718.

[IT16] Jun Inoue and Walid Taha. Reasoning about multi-stage programs. *Journal of Functional Programming*, 26(e22), 2016. doi:10.1017/S0956796816000253.

[JT93] Achim Jung and Jerzy Tiuryn. A new characterization of lambda definability. In *Proceedings of the International Conference on Typed Lambda Calculi and Applications*, pages 245–257. Springer, 1993. doi:10.1007/BFb0037110.

[KKS22] Ohad Kammar, Shin-ya Katsumata, and Philip Saville. Fully abstract models for effectful λ-calculi via category-theoretic logical relations. *Proceedings of the ACM on Programming Languages*, 6(POPL):1–28, 2022. doi:10.1145/3498705.

[Koc95] Anders Kock. Monads for which structures are adjoint to units. *Journal of Pure and Applied Algebra*, 104(1):41–59, 1995. doi:10.1016/0022-4049(94)00111-U.

[Kuč98] Jakov Kučan. Retraction approach to CPS transform. *Higher Order Symbol. Comput.*, 11(2):145–175, 1998. doi:10.1023/A:1010012532463.

[LD93] Julia L. Lawall and Olivier Danvy. Separating stages in the continuation-passing style transformation. In *Proceedings of the 20th ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages*, pages 124–136. ACM, 1993. doi:10.1145/158511.158613.

[Lev99] Paul Blain Levy. Call-by-push-value: A subsuming paradigm. In Jean-Yves Girard, editor, *Typed Lambda Calculi and Applications*, pages 228–243. Springer, 1999. doi:10.1007/3-540-48959-2_17.

[Lev03] Paul Blain Levy. Adjunction models for call-by-push-value with stacks. *Electronic Notes in Theoretical Computer Science*, 69:248–271, 2003. CTCS'02, Category Theory and Computer Science. doi:https://doi.org/10.1016/S1571-0661(04)80568-1.

[Lev06] Paul Blain Levy. Call-by-push-value: Decomposing call-by-value and call-by-name. *Higher-Order and Symbolic Computation*, 19(4):377–414, 2006. doi:10.1007/s10990-006-0480-6.

[MM19] Dylan McDermott and Alan Mycroft. Extended call-by-push-value: Reasoning about effectful programs and evaluation order. In Luís Caires, editor, *Programming Languages and Systems*, pages 235–262. Springer, 2019. doi:10.1007/978-3-030-17184-1_9.

[MM22] Dylan McDermott and Alan Mycroft. Galois connecting call-by-value and call-by-name. In Amy P. Felty, editor, *7th International Conference on Formal Structures for Computation and Deduction (FSCD 2022)*, volume 228 of *Leibniz International Proceedings in Informatics (LIPIcs)*, pages 32:1–32:19, Dagstuhl, Germany, 2022. Schloss Dagstuhl – Leibniz-Zentrum für Informatik. doi:10.4230/LIPIcs.FSCD.2022.32.

[Mog91] Eugenio Moggi. Notions of computation and monads. *Inf. Comput.*, 93(1):55–92, 1991. doi:10.1016/0890-5401(91)90052-4.

[MOTW95] John Maraist, Martin Odersky, David N. Turner, and Philip Wadler. Call-by-name, call-by-value, call-by-need, and the linear lambda calculus. In *Proceedings of the Eleventh Annual Mathematical Foundations of Programming Semantics Conference*, pages 370–392, 1995. doi:10.1016/S1571-0661(04)00022-2.

[MSS86] Austin Melton, David A. Schmidt, and George E. Strecker. Galois connections and computer science applications. In *Category Theory and Computer Programming*, pages 299–312. Springer, 1986. doi:10.1007/3-540-17162-2_130.

[MU22] Dylan McDermott and Tarmo Uustalu. What makes a strong monad? In *Proceedings Ninth Workshop on Mathematically Structured Functional Programming (to appear)*. Open Publishing Association, 2022.

[MW85] Albert R. Meyer and Mitchell Wand. Continuation semantics in typed lambda-calculi. In Rohit Parikh, editor, *Logics of Programs*, pages 219–224. Springer, 1985. doi:10.1007/3-540-15648-8_17.

[NL20] Max S. New and Daniel R. Licata. Call-by-name gradual type theory. *Logical Methods in Computer Science*, 16, 2020. doi:10.23638/LMCS-16(1:7)2020.

[NLA21] Max S. New, Daniel R. Licata, and Amal Ahmed. Gradual type theory. *Journal of Functional Programming*, 31, 2021. doi:10.1017/S0956796821000125.

[Par92] Michel Parigot. $\lambda\mu$-calculus: An algorithmic interpretation of classical natural deduction. In Andrei Voronkov, editor, *Logic Programming and Automated Reasoning*, pages 190–201. Springer, 1992. doi:10.1007/BFb0013061.

[Plo75] G. D. Plotkin. Call-by-name, call-by-value and the λ-calculus. *Theoretical Computer Science*, 1(2):125–159, 1975. doi:10.1016/0304-3975(75)90017-1.

[Rey74] John C. Reynolds. On the relation between direct and continuation semantics. In *Proceedings of the 2nd Colloquium on Automata, Languages and Programming*, pages 141–156. Springer, 1974. doi:10.1007/978-3-662-21545-6_10.

[Sel01] Peter Selinger. Control categories and duality: on the categorical semantics of the lambda-mu calculus. *Mathematical Structures in Computer Science*, 11(2):207–260, 2001. doi:10.1017/S096012950000311X.

[SF92] Amr Sabry and Matthias Felleisen. Reasoning about programs in continuation-passing style. In *Proceedings of the 1992 ACM Conference on LISP and Functional Programming*, pages 288–298. ACM, 1992. doi:10.1145/141471.141563.

[SPU22] José Espírito Santo, Luís Pinto, and Tarmo Uustalu. Plotkin's call-by-value λ-calculus as a modal calculus. *Journal of Logical and Algebraic Methods in Programming*, 2022. doi:10.1016/j.jlamp.2022.100775.

[SW96] Amr Sabry and Philip Wadler. A reflection on call-by-value. In *Proceedings of the First ACM SIGPLAN International Conference on Functional Programming*, pages 13–24. ACM, 1996. doi:10.1145/232627.232631.

[Wad03] Philip Wadler. Call-by-value is dual to call-by-name. In *Proceedings of the Eighth ACM SIGPLAN International Conference on Functional Programming*, pages 189–201. ACM, 2003. doi:10.1145/944705.944723.

[Win93] Glynn Winskel. *The formal semantics of programming languages: An introduction*. MIT Press, 1993. doi:10.7551/mitpress/3054.001.0001.

This work is licensed under the Creative Commons Attribution License. To view a copy of this license, visit https://creativecommons.org/licenses/by/4.0/ or send a letter to Creative Commons, 171 Second St, Suite 300, San Francisco, CA 94105, USA, or Eisenacher Strasse 2, 10777 Berlin, Germany