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

DYLAN MCDERMOTT \textsuperscript{a} AND ALAN MYCROFT \textsuperscript{b}

\textsuperscript{a} Reykjavik University, Iceland
\textit{e-mail address:} dylanm@ru.is

\textsuperscript{b} University of Cambridge, UK
\textit{e-mail address:} Alan.Mycroft@cl.cam.ac.uk

\begin{abstract}
We establish a general framework for reasoning about the relationship between call-by-value and call-by-name.

In languages with computational 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 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 computational 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 effects that imply these maps form a Galois connection. These properties hold for some computational 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 computational effects including divergence and nondeterminism.
\end{abstract}

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

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

\textit{Key words and phrases:} computational effect, evaluation order, call-by-push-value, categorical semantics.

\textsuperscript{*} This article is an extended version of [MM22].
• 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 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 \( V[e] \) and \( N[e] \), which respectively correspond to call-by-value and call-by-name. We study the relationship between the behaviour of \( V[e] \) and the behaviour of \( N[e] \) in a given program context.

The main obstacle is that \( V[e] \) and \( N[e] \) have different types. (The former has a “call-by-value type” \( F(V[\tau]) \) and the latter a “call-by-name type” \( N[\tau] \), 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 \( \lambda \)-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 \( V[e] \) is related in this sense to \( N[e] \). This is not the case in general. In the presence of arbitrary computational effects, we cannot expect to say anything useful about how the behaviour of \( V[e] \) relates to the behaviour of \( N[e] \). However, under certain conditions satisfied only for certain effects, \( V[e] \) is related to \( N[e] \). These conditions say roughly that we can discard, duplicate, and reorder 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 computational effects than in the first, saying informally that we can thunk 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 \( \preceq \) that captures the property we wish to show about programs, our reasoning principle gives conditions that imply \( M \preceq M' \), 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 \( \preceq \); each of these relations indicates the extent to which changing evaluation order affects the behaviour of the program. In the divergence example \( N \preceq N' \) is defined to mean termination of \( N \) implies termination of \( N' \) with the same result; in the other examples \( \preceq \) similarly mirrors the properties described informally above.
Rather than just considering some fixed collection of (allowable) effects, we work abstractly and identify properties of computational effects that enable us to relate call-by-value and call-by-name.

Our reasoning principle relies on the existence of some denotational model $\mathcal{M}$ of the computational effects, and we reason primarily inside $\mathcal{M}$. In the first step we in fact relate the behaviour of the call-by-value and call-by-name translations of terms within the given model $\mathcal{M}$. On the other hand, in the second step we are able to prove a result (our reasoning principle Theorem 6.2) in which the conclusion is independent of $\mathcal{M}$ (though Theorem 6.2 does assume the existence of a suitable $\mathcal{M}$, since the proof relies of the first step).

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 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 $\text{bool}$ (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 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 $\text{bool}$ (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.¹

The syntax of CBPV terms is stratified into two kinds: values \( V, W \) do not reduce, computations \( M, N \) might reduce (possibly with computational effects). The syntax of types is similarly stratified into value types \( A, B \) and computation types \( C, D \).

value types \( A, B \) ::= \text{unit} | A_1 \times A_2 | \text{bool} | U_C \\
computation types \( C, D \) ::= C_1 \times C_2 | A \to C | F A \\
values \( V, W \) ::= x | () | (V_1, V_2) | \text{true} | \text{false} | \text{thunk} M \\
computations \( M, N \) ::= \lambda \{1. M_1, 2. M_2\} | 1'M | 2'M \\
| \lambda x : A. M | V'M | \text{return} V | M \text{ to } x. N \\
| \text{match } V \text{ with } (x_1, x_2). M \\
| \text{if } V \text{ then } M_1 \text{ else } M_2 | \text{force } V

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

The value type \( U_C \) is the type of \textit{thunks} of computations of type \( C \). Elements of \( U_C \) are introduced using \textbf{thunk}: the value \textbf{thunk} \( M \) is the suspension of the computation term \( M \). The corresponding eliminator is \textbf{force}, which is the inverse of \textbf{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 \( 1'M \) and \( 2'M \). Computation types also include function types (where functions send values to computations). Function application is written \( V'M \), where \( V \) is the argument and \( M \) is the function to apply. The \textit{returner type} \( F A \) has as elements computations that return elements of the value type \( A \); these computations may have effects. Elements of \( F A \) are introduced by \textbf{return}; the computation \textbf{return} \( V \) immediately returns the value \( V \) (with no effects). Computations can be sequenced using \( M \text{ to } x. N \). 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 \( \textit{M} >>= \backslash x \rightarrow N \) in Haskell.) The syntax we give here does not include any method of introducing effects; we extend CBPV with some example computational 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 \( \text{to} \), 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 \( \Gamma \vdash c_M : C \) for computations. \textit{Typing contexts} \( \Gamma \) 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 \( \emptyset \) for the empty typing context, \( V : A \) as an abbreviation for \( \emptyset \vdash V : A \), and \( M : C \) as an abbreviation for \( \emptyset \vdash c_M : C \).

¹Our use of CBPV should not be regarded as essential. We use it here because it is known to capture call-by-value and call-by-name in a strong sense (see [Lev99]). It may be possible to replace CBPV with some other language that captures call-by-value and call-by-name, and obtain similar results to ours.
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 \textit{terminal computations}, which are the subset of computations with an introduction form on
the outside:

\[ R ::= \lambda\{1, M_1, 2, M_2\} | \lambda x : A. M | \text{return} V \]

We only evaluate closed, well-typed computations, so when we write \( M \Downarrow R \) we assume \( M : C \) for some \( C \) (this implies \( R : C \)). 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 \( \hat{i} M \), only the \( i \)th component of the pair \( M \) is evaluated. Since we have not yet included any way of forming impure computations, the semantics is deterministic and normalizing: given any \( M : C \), 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 \( M : \text{F bool} \). 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 \( \preceq \) on programs. For example, we could use

\[ M \preceq M' \text{ if and only if } \forall V : \text{bool}. (M \Downarrow \text{return} V) \Rightarrow (M' \Downarrow \text{return} V) \]

We could also use, for example, the total relation for \( \preceq \) (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 \( \preceq \), we define a contextual preorder \( M \preceq_{\text{ctx}} M' \) on arbitrary well-typed computations (in typing context \( \Gamma \)) by considering the behaviour of \( M \) and \( M' \) in programs as follows. A computation context \( E \) is a computation term, with a single hole \( \square \) where a computation term is expected. We write \( E[M] \) for the computation that results from replacing \( \square \) with \( M \) (which may capture some of the free variables of \( M \)). For example, if \( E \) is the computation context \( \text{N to} \ x. \square \) then \( E[\text{return} x] \) is the computation \( \text{N to} \ x. \text{return} x \), where \( x \) is captured. We use computation contexts to define \( \preceq_{\text{ctx}} \).

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

We sometimes omit \( \Gamma \), and write just \( M \preceq_{\text{ctx}} M' \) or \( M \equiv_{\text{ctx}} M' \).

### 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 in a strong sense (see the introduction of [Lev99] for a detailed discussion of this). 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 \( \tau \) and expressions \( e \):

\[
\tau ::= \text{unit} | \text{bool} | \tau_1 \times \tau_2 | \tau \to \tau' \\
e ::= x | () | (e_1, e_2) | \text{fst} e | \text{snd} e | \text{true} | \text{false} | \text{if} e_0 \text{ then } e_1 \text{ else } e_2 | \lambda x : \tau. e | e e'
\]

\(2\)We do not actually need to assume that \( \preceq \) 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 \( \preceq \) is not a preorder.
We include two base types \texttt{unit} and \texttt{bool} to be used in examples.\footnote{Unlike in Levy [Lev99], we do not include general sum types, only \texttt{bool}. We expect that including arbitrary sum types would complicate Section 4, because it is difficult to extend logical relations of varying arity with sums. The difficulty, and techniques for dealing with it, are discussed e.g. in [AS19, FS99, Kat08].} 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 $\tau$ is mapped to a CBPV value type $V(\tau)$ that contains the results of call-by-value computations. For call-by-name, $\tau$ is translated to a computation type $N(\tau)$, which contains the computations themselves. Products in call-by-value use the value-type products of CBPV (which means they are necessarily \textit{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 \texttt{unit} as a nullary product of computation types. We instead treat \texttt{unit} as a base type, so that effects can happen at type \texttt{unit}, which matches typical functional languages.) Functions under the call-by-value translation accept values of type $V(\tau)$ as arguments; arguments are evaluated before being passed to the function. Under the call-by-name translation, functions accept thunks of computations as arguments; instead of evaluating them, arguments are thunked before passing them to call-by-name functions. Source-language typing contexts $\Gamma$ are translated to CBPV typing contexts $V(\Gamma)$ and $N(\Gamma)$. 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 $V(e)$ and $N(e)$. 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, computational effects occur only at the base types \texttt{unit} and \texttt{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 $V(e)$ and $V(e')$ 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) effects as examples throughout the paper.

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

$$M \triangleleft_{\text{pure}} M' \text{ if and only if } \exists V : \text{bool}. \ (M \triangleright \text{return} V) \land (M' \triangleright \text{return} V)$$

In other words, $M$ and $M'$ both evaluate to the same result $V$. Since evaluation is deterministic, $V$ is necessarily unique. The contextual preorder $M \triangleleft_{\text{ctx}} M'$ means if we construct two programs by wrapping $M$ and $M'$ in the same computation context, then
The figure shows the translations from the source language into CBPV. The left side describes call-by-value translation, while the right side describes call-by-name translation.

Call-by-value translation involves:
- **Type**: \( \tau \rightarrow \text{value type } V(\tau) \)
- **Unit**: \( \text{unit} \rightarrow \text{unit} \)
- **Product**: \( \tau_1 \times \tau_2 \rightarrow V(\tau_1) \times V(\tau_2) \)
- **Boolean**: \( \text{bool} \rightarrow \text{bool} \)
- **Function**: \( \tau \rightarrow \tau' \rightarrow U(V(\tau) \rightarrow F(V(\tau'))) \)

Call-by-name translation involves:
- **Type**: \( \tau \rightarrow \text{computation type } N(\tau) \)
- **Unit**: \( F \text{ unit} \)
- **Product**: \( \tau_1 \times \tau_2 \rightarrow N(\tau_1) \times N(\tau_2) \)
- **Boolean**: \( F \text{ bool} \)
- **Function**: \( \tau \rightarrow \tau' \rightarrow (U(N(\tau)) \rightarrow N(\tau')) \)

The figure is labeled as Figure 3. Translations from the source language into CBPV.

(A) Call-by-value translation \( V(\_\_\_) \)

(B) Call-by-name translation \( N(\_\_\_) \)
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 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 $\preceq_{\text{ctx}}$ as it is defined in our no-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 $\text{rec } x : \mathbb{C} . M$, and correspondingly extend the type system and operational semantics with the following rules:

$$
\frac{\Gamma, x : \mathbb{C} \vdash_c M : C}{\Gamma \vdash_c \text{rec } x : \mathbb{C} . M : C}
\quad
\frac{M \equiv \text{thunk} \left( \text{rec } x : \mathbb{C} . M \right) \downarrow R}{\text{rec } x : \mathbb{C} . M \downarrow R}
$$

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:

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

\[
\begin{align*}
\mathcal{V}(\text{rec } f : \tau \to \tau'. \lambda x. e) &= \text{return thunk}(\text{rec } f : \mathbb{U}(\mathcal{V}(\|\tau\|)) \to \mathbb{F}(\mathcal{V}(\|\tau'\|))). \lambda x : \mathcal{V}(\|\tau\|). \mathcal{V}(\|e\|) \\
\mathcal{N}(\text{rec } f : \tau \to \tau'. \lambda x. e) &= \text{rec } f : \mathbb{U}(\mathcal{N}(\|\mathcal{N}(\|\tau\|)) \to \mathcal{N}(\|\tau'\|)). \lambda x : \mathcal{U}(\|\mathcal{N}(\|\tau\|)) . \mathcal{U}(\|e\|)
\end{align*}
\]

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 $\Omega_\tau = ((\text{rec } f : \text{bool} \to \tau. \lambda x. f x) \text{ false}) : \tau$ enables us to distinguish between call-by-value and call-by-name: $(\lambda x : \tau. \text{true}) \Omega_\tau$ diverges in call-by-value but not in call-by-name. In particular, we have $\mathcal{N}(((\lambda x : \tau. \text{true}) \Omega_\tau) \downarrow \text{return } \text{true}$, but there is no $R$ such that $\mathcal{V}((\lambda x : \tau. \text{true}) \Omega_\tau) \downarrow R$.

For this example, we define the program relation $\preceq_{\text{div}}$ by

$$
M \preceq_{\text{div}} M' \text{ if and only if } \forall V : \text{bool}. (M \downarrow \text{return } V) \Rightarrow (M' \downarrow \text{return } V)
$$

so that $M \preceq_{\text{ctx}} M'$ 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 $\text{fail}_C$ for nullary nondeterministic choice and $M \text{ or } N$ for binary nondeterministic choice between computations; the typing and evaluation rules are standard:

\[
\begin{align*}
\frac{\Gamma \vdash_c \text{fail}_C : C}{\quad}{
\frac{\Gamma, M : C \vdash \quad \Gamma, N : C \vdash \quad M \downarrow R \quad N \downarrow R}{\quad \Gamma \vdash_c \text{fail}_C : C \quad \Gamma \vdash_c \text{or } M \downarrow R \quad \text{or } N \downarrow R}
\end{align*}
\]
(There is no \( R \) such that \( \text{fail}_C \downarrow R \).) The computation \( \text{fail}_C \) is the unit for \( \text{or} \), so \( \text{fail}_C \text{ or } M \) and \( M \text{ or } \text{fail}_C \) have the same behaviour as \( M \). For each closed computation \( M : F \mathcal{A} \) there might be zero, one or several values \( V : A \) such that \( M \downarrow \text{return } V \).

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

\[
\begin{align*}
\text{e} &::= \ldots | \text{fail}_r | e \text{ or } e' \\
\Gamma \vdash \text{fail}_r : \tau & \quad \Gamma \vdash e : \tau \quad \Gamma \vdash e' : \tau
\end{align*}
\]

\[V\langle \text{fail}_r \rangle = \text{fail}_F\langle V\langle r \rangle \rangle \]
\[V\langle e \text{ or } e' \rangle = V\langle e \rangle \text{ or } V\langle e' \rangle \]
\[\mathcal{N}\langle \text{fail}_r \rangle = \text{fail}_\mathcal{N}\langle V\langle r \rangle \rangle \]
\[\mathcal{N}\langle e \text{ or } e' \rangle = \mathcal{N}\langle e \rangle \text{ or } \mathcal{N}\langle e' \rangle \]

As an example, evaluating the expression \( e = (\lambda x. \text{if } x \text{ then } x \text{ else true})(\text{true or false}) \) under call-by-value necessarily results in \( \text{true} \), but under call-by-name we can also get \( \text{false} \).

(We have \( V\langle e \rangle \nvdash \text{return false} \) but \( \mathcal{N}\langle e \rangle \downarrow \text{return false} \).)

For nondeterminism, we define \( \preceq_{\text{nd}} \) in the same way as our divergence example:

\[M \preceq_{\text{nd}} M' \text{ if and only if } \forall b \in \{\text{true, false}\}. \exists V : \text{bool}. (M \downarrow \text{return } V) \implies (M' \downarrow \text{return } V)\]

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 \( \text{true} \) or \( \text{false} \). 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 \( \text{bool} \), 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 \( \text{get} \). This gets the value of the state, producing either \( \text{true} \) or \( \text{false} \).

\[\Gamma \vdash e : \text{Fbool} \]

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 \( b \in \{\text{true, false}\} \). The rules are those of Figure 2 (with the subscript \( b \) added), plus

\[\text{get} \downarrow_{\text{true}} \text{return true} \quad \text{get} \downarrow_{\text{false}} \text{return false}\]

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

\[
\begin{align*}
\text{e} &::= \ldots | \text{get} \\
\Gamma \vdash \text{get} : \text{bool} & \quad V\langle \text{get} \rangle = \mathcal{N}\langle \text{get} \rangle = \text{get}
\end{align*}
\]

We define the program relation \( \preceq_{\text{get}} \) as follows:

\[M \preceq_{\text{get}} M' \text{ if and only if } \forall b \in \{\text{true, false}\}. \exists V : \text{bool}. (M \downarrow_b \text{return } V) \land (M' \downarrow_b \text{return } V)\]
3. Order-enriched denotational semantics

We give a denotational semantics for CBPV, which we use to prove instances of $\preceq_{\text{ctx}}$. Since $\preceq_{\text{ctx}}$ 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 $M \preceq_{\text{ctx}} N$. 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 $\mathcal{C}$ is an ordinary category, together with a partial order $\sqsubseteq$ on each hom-set $\mathcal{C}(X,Y)$, such that composition is monotone.

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

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

| Poset-category $\mathcal{C}$ | Objects $X \in |\mathcal{C}|$ | Morphisms $f : X \to Y$ | Order $f \sqsubseteq f'$ |
|-----------------------------|-------------------------------|--------------------------|-------------------------|
| Set                         | sets                          | functions                | equality                |
| Poset                       | posets                        | monotone functions       | pointwise               |
| $\omega\text{Cpo}$         | $\omega\text{cpos}$           | $\omega$-continuous      | functions pointwise     |

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

Let $\mathcal{C}$ be a Poset-category. We say that $\mathcal{C}$ is cartesian when its underlying category has a terminal object 1 and binary products $X_1 \times X_2$, such that the pairing functions $\langle -, - \rangle : \mathcal{C}(W,X_1) \times \mathcal{C}(W,X_2) \to \mathcal{C}(W,X_1 \times X_2)$ 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 $\text{assoc}_{X_1,X_2,X_3} : (X_1 \times X_2) \times X_3 \to X_1 \times (X_2 \times X_3)$. We say that $\mathcal{C}$ is cartesian closed when it is cartesian and its underlying category has exponentials $X \Rightarrow Y$ for which the currying functions $\Lambda : \mathcal{C}(W \times X,Y) \to \mathcal{C}(W,X \Rightarrow Y)$ are monotone. We write $\text{ev}_{X,Y}$ for the evaluation morphism $\Lambda^{-1} \text{id} : (X \Rightarrow Y) \times X \to Y$.

Binary coproducts in $\mathcal{C}$ are just binary coproducts in the underlying ordinary category, except that the copairing functions $[\cdot,\cdot] : \mathcal{C}(X_1,W) \times \mathcal{C}(X_2,W) \to \mathcal{C}(X_1 + X_2,W)$ are required to be monotone. We write $\text{inl} : X_1 \to X_1 + X_2$ and $\text{inr} : X_2 \to X_1 + X_2$ for the coprojections. The Poset-categories Set, Poset, and $\omega\text{Cpo}$ are all cartesian closed, and have binary coproducts given by disjoint union.
Above we ask for monotonicity of the bijections
\[
(-,-) : \mathbf{C}(W, X_1) \times \mathbf{C}(W, X_2) \to \mathbf{C}(W, X_1 \times X_2)
\]
\[
\Lambda : \mathbf{C}(W \times X, Y) \to \mathbf{C}(W, X \Rightarrow Y)
\]
\[
[-,-] : \mathbf{C}(X_1, W) \times \mathbf{C}(X_2, W) \to \mathbf{C}(X_1 + X_2, W)
\]
We do not need to require monotonicity of their inverses explicitly, because this holds automatically. In particular, the uncurrying functions \(\Lambda^{-1} : \mathbf{C}(W, X \Rightarrow Y) \to \mathbf{C}(W \times X, Y)\) are monotone because \(\Lambda^{-1} f = ev_{X,Y} \circ (f \times id_X)\), and \(\circ\) and \(\times\) are both monotone.

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 \(\text{Poset}\)-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 \(\text{Poset}\)-monad).** Let \(\mathbf{C}\) be a cartesian \(\text{Poset}\)-category. A strong \(\text{Poset}\)-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 \((-)^{+\times\Box} : \mathbf{C}(W \times X, TY) \to \mathbf{C}(W \times TX, TY)\) (Kleisli extension)
  - for each \(W, X, Y \in |\mathbf{C}|\).

These are required to satisfy the following four laws.\(^4\)

- **Naturality of extension in \(W\):**
  \[
  f^{+\times\Box} \circ (w \times id_{TX}) = (f \circ (w \times id_X))^{+\times\Box}
  \]
  for all \(f : W \times X \to TY\) and \(w : W' \to W\).
- **Left unit:**
  \[
  f^{+\times\Box} \circ (id_W \times \eta_X) = f
  \]
  for all \(f : W \times X \to TY\).
- **Right unit:**
  \[
  (\eta_X \circ \pi_2)^{1\times\Box} = \pi_2
  \]
  for all \(X \in |\mathbf{C}|\).
- **Associativity:**
  \[
  (g^{+\times\Box} \circ (id_{W'} \times f) \circ \text{assoc})^{+\times(W'\times W)\times\Box} = g^{+\times\Box} \circ (id_{W'} \times f^{+\times\Box}) \circ \text{assoc}
  \]
  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:

\[
 f^\dagger \circ \eta_X = f \quad \eta_X^\dagger = id_X \quad (g^\dagger \circ f)^\dagger = g^\dagger \circ f^\dagger
\]

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 \(\text{Poset}\)-functor: the mapping \(f \mapsto Tf\) is monotone, and

\(^4\)The conference version [MM22] of this paper incorrectly omits naturality in \(W\) from the definition of strong \(\text{Poset}\)-monad and from the definition of Eilenberg–Moore algebra. (Naturality in \(W\) is required in [MU22, Definition 4.1].)
We also define two natural transformations for sequencing of computations:

\[ T f \circ \eta_X = \eta_Y \circ f \quad \text{(Tg \circ f)} \]

where \( \square W \times \square \) preserves identities and composition. The definition of \( T \) on morphisms also ensures that unit and Kleisli extension of \( T \) satisfy the following naturality laws:

\[ T f \circ \eta_X = \eta_Y \circ f \quad \text{(Tg \circ f)} \]

In the notation \( f^{W \times \square} : W \times TX \to TY \), the square \( \square \) 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^{\square \times W} = (f \circ (\pi_2, \pi_1))^{\square \times W} \circ (\pi_2, \pi_1) : TX \times W \to TY \quad \text{(where \( f : X \times W \to TY \))} \]

We also define two natural transformations for sequencing of computations: \( \text{seq}_L^T \) for left-to-right and \( \text{seq}_R^T \) for right-to-left, as follows.

\[
\text{seq}_L^{X_1, X_2} = (\eta_{X_1 \times X_2}^{\square \times X_2})^{\square \times TX_2} : TX_1 \times TX_2 \to T(X_1 \times X_2)
\]

\[
\text{seq}_R^{X_1, X_2} = (\eta_{X_1 \times X_2}^{\square \times X_2})^{\square \times TX_1} : TX_1 \times TX_2 \to T(X_1 \times X_2)
\]

We further define an effectful pairing operation \( \langle - , - \rangle \):

\[
\langle f_1 , f_2 \rangle = \text{seq}_L^{X_1, X_2} \circ (f_1 , f_2) : W \to T(X_1 \times X_2)
\]

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 \( C \). A \( T \)-algebra \( Z = (Z, (\langle - , - \rangle)^T) \) is a pair of:

- an object \( Z \in |C| \) (the carrier);
- a monotone function \((\langle - , - \rangle)^T : C(W \times X, Z) \to C(W \times TX, Z) \) (the extension operator)

for each \( W,X \in |C| \).

These are required to satisfy the following three laws.

- Naturality in \( W \):
  \[ f^{W \times \square} \circ (w \times id_{TX}) = (f \circ (w \times id_X))^{W' \times \square} \]
  for all \( f : W \times X \to Z \) and \( w : W' \to W \).

- Left unit:
  \[ f^{W \times \square} \circ (id_W \times \eta_X) = f \]
  for all \( f : W \times X \to Z \).

- Associativity:
  \[ (g^{W' \times \square} \circ (id_W \times f) \circ assoc)^{W \times W' \times \square} = g^{W' \times \square} \circ (id_W \times f^{W \times \square}) \circ assoc \]
  for all \( f : W \times X \to TY \) and \( g : W' \times Y \to Z \).

For each \( T \)-algebra \( Z \), we write \( U_T Z \) for the carrier \( Z \in |C| \).

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 \((\langle - , - \rangle)^T : C(X, Z) \to C(TX, Z) \). We also have an extension operator with reversed products, written \((\langle - , - \rangle)^{\square \times W} : C(X \times W, Z) \to C(TX \times W, Z) \).

The following constructions of algebras are standard.

**Definition 3.5.** Let \( T \) be a strong Poset-monad on a cartesian closed Poset-category \( C \).
The free \( T \)-algebra \( F_T X \) on an object \( X \in |C| \) has carrier \( TX \); the extension operator is Kleisli extension \((-)^\dagger\).

- If \( Z_1 \) and \( Z_2 \) are \( T \)-algebras, then their product \( Z_1 \times Z_2 \) is the \( T \)-algebra with carrier \( Z_1 \times Z_2 \), and extension operator
  \[
  f^{1 \times \Box} = ((\pi_1 \circ f)^{1 \times \Box}, (\pi_2 \circ f)^{1 \times \Box})
  \]

- If \( Y \in |C| \) and \( Z \) is a \( T \)-algebra, then their power \( Y \Rightarrow Z \) is the \( T \)-algebra with carrier \( Y \Rightarrow Z \) and extension operator
  \[
  f^{1 \times \Box} = \Lambda((\Lambda^{-1} f \circ \beta_{W,Y,X})^{1 \times (W \times Y) \times \Box} \circ \beta_{W,TX,Y})
  \]
  where \( \beta_{X_1,X_2,X_3} = \langle (\pi_1 \circ \pi_1, \pi_2 \circ \pi_1) : (X_1 \times X_2) \times X_3 \rightarrow (X_1 \times X_3) \times X_2 \rangle \).

We use these constructions to interpret CBPV computation types: returner types \( \Gamma \vdash c : \Lambda \) are interpreted as free \( T \)-algebras, product types \( \Gamma \vdash C_1 \times C_2 \) are interpreted as product \( T \)-algebras, and function types \( \Gamma \vdash A \rightarrow C \) 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 \( M = (C, T) \) of CBPV consists of

- a cartesian closed \textbf{Poset}-category \( C \) that admits the coproduct \( 2 = 1 + 1 \);
- a strong \textbf{Poset}-monad \( T \) on \( C \).

Given a model \( M = (C, T) \), the interpretation \([\cdot] \) of CBPV is defined in Figure 4. Value types \( A \) are interpreted as objects \([A] \in |C| \), while computation types \( C \) are interpreted as \( T \)-algebras. The value type \( UC \) is interpreted as the carrier \( UT[C] \) of the \( T \)-algebra \([C] \). Typing contexts \( \Gamma \) are interpreted as objects \([\Gamma] \in C \) using the cartesian structure of \( C \); if \( \langle x : A \rangle \in \Gamma \) then we write \( \pi_x \) for the corresponding projection \([\Gamma] \rightarrow [A] \). Values \( \Gamma \vdash V : A \) (respectively computations \( \Gamma \vdash M : C \)) are interpreted as morphisms \([\Gamma \vdash V : A] \) (resp. \( [\Gamma \vdash M : C] \)) in \( C \); we often omit the typing context and type when writing these.

Programs \( \phi \vdash M : \text{bool} \) are therefore interpreted as morphisms \([M] : 1 \rightarrow T2 \). To interpret if, we use the fact that, since \( C \) is cartesian closed, products distribute over the coproduct \( 2 = 1 + 1 \). This means that for every \( W \in |C| \), the coproduct \( W + W \) also exists in \( C \), and the canonical morphism

\[
W + W \xrightarrow{[id_W, inl(W), \text{inr}(W)]} W 	imes 2
\]

has an inverse \( \text{dist}_W : W \times 2 \rightarrow W + W \).

By composing the semantics of CBPV with the two translations of the source language, we obtain a call-by-value semantics \( \mathcal{V}[\cdot] = [\mathcal{V}[\cdot]] \) and a call-by-name semantics \( \mathcal{N}[\cdot] = [\mathcal{N}[\cdot]] \) 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 \( \preceq \) if for all computations \( \Gamma \vdash_c M : C \) and \( \Gamma \vdash_c M' : C \) we have

\[
[\Gamma \vdash_c M : C] \sqsubseteq [\Gamma \vdash_c M' : C] \quad \Rightarrow \quad M \preceq_{\text{ctx}} M'
\]
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 ≼; the proof in each case is a standard logical relations argument (e.g. [Win93]).

Example 3.8. For CBPV with no effects, we use $C = \text{Set}$. The strong Poset-monad $T$ is the identity on $\text{Set}$. Each $T$-algebra $Z$ is completely determined by its carrier $Z$; the extension operator $(-)^{\times \Box}_T : \text{Set}(W \times X, Z) \to \text{Set}(W \times X, Z)$ is necessarily the identity. The interpretation $[M]$ of a closed computation $M : F\text{bool}$ is just an element of 2.

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

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

A $T$-algebra $Z$ is equivalently an $\omega\text{Cpo}$ $Z$ with a least element $\perp \in Z$. The extension operator is completely determined once the carrier is fixed; it is analogous to $(-)^{\times \Box}$. 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 $\omega$-continuous functions ordered pointwise. Hence $Z_1 \times Z_2$ has a least element $(\perp, \perp)$ (so forms a $T$-algebra) whenever $Z_1$ and $Z_2$ have least
\begin{figure}[h]
\begin{align*}
\text{type } \tau & \mapsto \text{object } \mathcal{V}[	au] \in \mathcal{C} \\
\text{unit} & \mapsto 1 \\
\tau_1 \times \tau_2 & \mapsto \mathcal{V}[[\tau_1] \times \mathcal{V}[[\tau_2]] \\
\text{bool} & \mapsto 2 \\
\tau \to \tau' & \mapsto \mathcal{V}[[\tau] \Rightarrow T(\mathcal{V}[[\tau']])
\end{align*}
\end{figure}

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

\begin{align*}
x & \mapsto \pi_x \\
() & \mapsto \eta_1 \circ \langle \mathcal{V}[[\Gamma]] \\
(e_1,e_2) & \mapsto \langle \mathcal{V}[[e_1]],\mathcal{V}[[e_2]] \rangle \\
\text{fst } e & \mapsto T\pi_1 \circ \mathcal{V}[e] \\
\text{snd } e & \mapsto T\pi_2 \circ \mathcal{V}[e] \\
\text{true} & \mapsto \eta_2 \circ \text{inl} \circ \langle \mathcal{V}[[\Gamma]] \\
\text{false} & \mapsto \eta_2 \circ \text{inr} \circ \langle \mathcal{V}[[\Gamma]] \\
\text{if } e_0 \text{ then } e_1 \text{ else } e_2 & \mapsto \langle \mathcal{V}[e_1],\mathcal{V}[e_2] \rangle \circ \text{dist}^{\mathcal{V}[[\Gamma]] \times \mathcal{V}[[\Gamma]]} \circ \langle \text{id}_{\mathcal{V}[[\Gamma]]},\mathcal{V}[e_0] \rangle \\
\lambda x : \tau. e & \mapsto \Lambda(\mathcal{V}[e]) \\
\text{ee}' & \mapsto \text{ev} \circ \langle \mathcal{V}[e],\mathcal{V}[e'] \rangle
\end{align*}

(\Lambda) \text{ Call-by-value semantics } \mathcal{V}[-]

\begin{figure}[h]
\begin{align*}
\text{type } \tau & \mapsto \text{T-algebra } \mathcal{N}[[\tau]] \\
\text{unit} & \mapsto F_T 1 \\
\tau_1 \times \tau_2 & \mapsto \mathcal{N}[[\tau_1]] \times \mathcal{N}[[\tau_2]] \\
\text{bool} & \mapsto F_T 2 \\
\tau \to \tau' & \mapsto (U_T(\mathcal{N}[[\tau]])) \Rightarrow \mathcal{N}[[\tau']]
\end{align*}
\end{figure}

expression \( \Gamma \vdash e : \tau \mapsto \text{morphism } \mathcal{N}[e] : \mathcal{N}[[\Gamma] \to U_T(\mathcal{N}[[\tau]]) \text{ in } \mathcal{C} \)

\begin{align*}
x & \mapsto \pi_x \\
() & \mapsto \eta_1 \circ \langle \mathcal{N}[[\Gamma]] \\
(e_1,e_2) & \mapsto \langle \mathcal{N}[e_1],\mathcal{N}[e_2] \rangle \\
\text{fst } e & \mapsto \pi_1 \circ \mathcal{N}[e] \\
\text{snd } e & \mapsto \pi_2 \circ \mathcal{N}[e] \\
\text{true} & \mapsto \eta_2 \circ \text{inl} \circ \langle \mathcal{N}[[\Gamma]] \\
\text{false} & \mapsto \eta_2 \circ \text{inr} \circ \langle \mathcal{N}[[\Gamma]] \\
\text{if } e_0 \text{ then } e_1 \text{ else } e_2 & \mapsto \langle \mathcal{N}[e_1],\mathcal{N}[e_2] \rangle \circ \text{dist}^{\mathcal{N}[[\Gamma]] \times \mathcal{N}[[\Gamma]]} \circ \langle \text{id}_{\mathcal{N}[[\Gamma]]},\mathcal{N}[e_0] \rangle \\
\lambda x : \tau. e & \mapsto \Lambda(\mathcal{N}[e]) \\
\text{ee}' & \mapsto \text{ev} \circ \langle \mathcal{N}[e],\mathcal{N}[e'] \rangle
\end{align*}

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

\textbf{Figure 5.} Denotational semantics of call-by-value and call-by-name
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 $\omega$-continuous function $f : Z \to Z$ has a least fixed point $\text{fix } f = \bigsqcup_{n \in \mathbb{N}} f^n \bot \in Z$. These enable us to interpret recursive computations, by defining

$$
\text{rec } x : \mathcal{F} C, \mathcal{M} \rho = \text{fix}(\lambda x. [\mathcal{M}](\rho, x)).
$$

The interpretation $[\mathcal{M}]$ of a closed computation $\mathcal{M} : \mathbb{F}\text{bool}$ is either $\bot$ (signifying divergence), or one of the two elements of $2$.

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

$$
TX = (\{\downarrow S' \mid S' \in \mathcal{P}_{\text{fin}} X\}, \subseteq) \quad \eta_X x = \downarrow\{x\} \quad f^{T[\sqcap]}(w, S) = \bigcup_{x \in S} f(w, x)
$$

where $\mathcal{P}_{\text{fin}} X$ is the set of finite subsets of $X$, and $\downarrow S' = \{x \in X \mid \exists x' \in S'. x \subseteq x'\}$ 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 $f^{T[\sqcap]}(w, S) = \bigcup_{x \in S} f(w, x)$. (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 $\bigcup_i (z_i, z'_i) = (\bigcup_i z_i, \bigcup_i z'_i)$. The power $Y \Rightarrow Z$ is the set of monotone functions ordered pointwise, with joins given by $(\bigcup_i f_i) x = \bigcup_i (f_i x)$.

We interpret nondeterministic computations using nullary and binary joins:

$$
\text{[fail]} \rho = \bot \quad [\text{M or } N] \rho = [\text{M}] \rho \sqcup [N] \rho
$$

The interpretation $[\mathcal{M}]$ of a closed computation $\mathcal{M} : \mathbb{F}\text{bool}$ is one of the four subsets of $2$.

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

$$
TX = (2 \Rightarrow X) \quad \eta_X x = \lambda b. x \quad f^{T[\sqcap]}(w, t) = \lambda b. f(w, t b) b
$$

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

$$
[\text{get}] \rho = \lambda b. b
$$

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 $\propto$ (Definition 4.1) that compare the observable behaviour of a denotation of call-by-value type with a denotation of call-by-name type. The main result of this section is that, under certain conditions on computational effects, we have

$$
\mathcal{V}[\mathcal{e}] \propto \mathcal{N}[\mathcal{e}]
$$

for all $\Gamma \vdash e : \tau$ (Theorem 4.7). Here we work with the denotational semantics, instead of with the syntax directly, so the relations $\propto$ are defined with respect to a fixed model $\mathcal{M}$ that we assume to be given. There is one relation $\propto$ for each source-language context $\Gamma$ and type $\tau$:

$$
g^v \propto g^n \text{ where } g^v : \mathcal{V}[\Gamma] \to T(\mathcal{V}[\tau]), \ g^n : \mathcal{N}[\Gamma] \to U_T(\mathcal{N}[\tau])
$$

To define $\propto$, we first give 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 \( U_T(\mathcal{N}[\tau]) \). Here by element we mean generalized element, so \( f^v \) and \( f^n \) are morphisms

\[
f^v : W \rightarrow T(\mathcal{V}[\tau]) \quad f^n : W \rightarrow U_T(\mathcal{N}[\tau])
\]

from some \( W \). The definition of \( \mathcal{R}[\tau]_W \) is in the style of a logical relation, by induction on the type \( \tau \). The cases are listed in Figure 6. Informally, we have the following.

- For \( \tau = \text{unit} \) and \( \tau = \text{bool} \), we compare \( f^v \) and \( f^n \) directly using the order relation \( \subseteq \) on morphisms in \( C \). (We can do this because \( T(\mathcal{V}[\tau]) = U_T(\mathcal{N}[\tau]) \).)
- 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 \( \text{fst} \) and \( \text{snd} \).
- For a function type \( \tau \rightarrow \tau' \), 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 \( w : W' \rightarrow W \) 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 \( w : W' \rightarrow W \), as in Lemma 4.5(1) below.)

We define \( \kappa \) in terms of \( \mathcal{R}[\cdot] \). 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 \( f_i : W \rightarrow [A_i] \) for each \( i \leq k \), we obtain a morphism \( \langle f_i \rangle_i : W \rightarrow [\Gamma] \), by iterated pairing. Given instead a morphism \( f_i : W \rightarrow T[A_i] \) for each \( i \), we obtain a morphism \( \langle f_i \rangle_i : W \rightarrow T[\Gamma] \) by iterated pairing and left-to-right evaluation:

\[
\langle f_i \rangle_{i \leq k} = \begin{cases} 
\langle \rangle_W & \text{if } k = 0 \\
\langle \langle f_i \rangle_{i \leq (k-1)}, f_k \rangle & \text{if } k > 0
\end{cases}
\]

\[
\langle f_i \rangle_{i \leq k} = \begin{cases} 
\eta_{\Gamma} \circ \langle \rangle_W & \text{if } k = 0 \\
\langle \langle f_i \rangle_{i \leq (k-1)}, f_k \rangle & \text{if } k > 0
\end{cases}
\]

**Definition 4.1.** Let \( M \) be a CBPV model, and let

\[
g^v : \mathcal{V}[\Gamma] \rightarrow T(\mathcal{V}[\tau']) \\
g^n : \mathcal{N}[\Gamma] \rightarrow U_T(\mathcal{N}[\tau'])
\]

be morphisms, where \( \Gamma = x_1 : \tau_1, \ldots, x_k : \tau_k \). We write

\[
g^v \propto g^n
\]

when, for all objects \( W \in |C| \) and families of morphisms

\[
(f_i^v : W \rightarrow T(\mathcal{V}[\tau_i]))_i \\
(f_i^n : W \rightarrow U_T(\mathcal{N}[\tau_i]))_i
\]
we have
\[ f_{i_1}^n \mathcal{R}[\tau_1]_W f_{i_2}^n \land \cdots \land f_{i_k}^n \mathcal{R}[\tau_k]_W f_{i_l}^n \Rightarrow (g^{n^1} \circ \langle f_{i_1}^n \rangle_i) \mathcal{R}[\tau]_W (g^n \circ \langle f_{i_k}^n \rangle_i) \]

As we mention above, our goal is relate the observable behaviour of \(V[e]\) to the observable behaviour of \(N[e]\). Precisely, we want to prove \(V[e] \mathord{\triangleright} N[e]\). 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
\[ e = (\lambda x : \text{bool}.()) : \text{bool} \to \text{unit} \]
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
\[ e = (\lambda x : \text{bool}. \text{if } x \text{ then } x \text{ else } x) : \text{bool} \to \text{bool} \]
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
\[ e = (\lambda x : \text{bool}. \lambda y : \text{bool}. \text{if } y \text{ then } x \text{ else } x) : \text{bool} \to \text{bool} \to \text{bool} \]
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
\[ T(\langle \rangle_Y) \circ f \sqsubseteq \eta_1 \circ \langle \rangle_X : X \to T1 \]
• **lax copyable** when
\[ T(\langle id_Y, id_Y \rangle) \circ f \sqsubseteq \text{seq}^L_{Y, Y} \circ \langle f, f \rangle : X \to T(Y \times Y) \]
equivalently, when
\[ T(\langle id_Y, id_Y \rangle) \circ f \sqsubseteq \text{seq}^R_{Y, Y} \circ \langle f, f \rangle : X \to T(Y \times Y) \]
• **lax central** when
\[ \text{seq}^L_{Y, W} \circ (f \times id_{TW}) \sqsubseteq \text{seq}^R_{Y, W} \circ (f \times id_{TW}) : X \times TW \to T(Y \times W) \]
for all \(W \in |C|\).

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 \(\Gamma \vdash_c M : F A\) be a CBPV computation. The following hold for every CBPV model that is adequate with respect to a program relation \(\preceq\).
• If \([M]\) is lax discardable, then
\[
M \text{ to } x. \text{return}(\) \lessdot_{\text{ctx}} \text{return}(\)
\]
• If \([M]\) is lax copyable, then
\[
M \text{ to } x. \text{return}(x, x) \lessdot_{\text{ctx}} M \text{ to } x_1. M \text{ to } x_2. \text{return}(x_1, x_2)
\]
• If \([M]\) is lax central, then
\[
M \text{ to } x. \text{force} z \text{ to } y. \text{return}(x, y) \lessdot_{\text{ctx}}^{\Gamma, z:U(\mathcal{FB})} \text{force} z \text{ to } y. M \text{ to } x. \text{return}(x, y)
\]

**Proof.** Since we assume adequacy, in each case we can reason inside the model.
• If \([M]\) is lax discardable, then
\[
[M \text{ to } x. \text{return}(\)] = T(\[\alpha][A] \circ [M] \\
\subseteq \eta_1 \circ (\[\alpha][\mathcal{R}]) \\
= [\text{return}(\)]
\]
• If \([M]\) is lax copyable, then
\[
[M \text{ to } x. \text{return}(x, x)] = T(id_{[\mathcal{A}], [\mathcal{A}]}) \circ [M] \\
\subseteq \text{seq}^L_{[\mathcal{A}], [\mathcal{A}]} \circ ([M], [M]) \\
= [M \text{ to } x_1. M \text{ to } x_2. \text{return}(x_1, x_2)]
\]
• If \([M]\) is lax central, then
\[
[M \text{ to } x. \text{force} z \text{ to } y. \text{return}(x, y)] = \text{seq}^R_{[\mathcal{A}], [\mathcal{B}]} \circ ([M] \times id_{T[B]}) \\
\subseteq \text{seq}^R_{[\mathcal{A}], [\mathcal{B}]} \circ ([M] \times id_{T[B]}) \\
= [\text{force} z \text{ to } y. M \text{ to } x. \text{return}(x, y)] 
\]

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 effects), the relations \(\mathcal{R}[\tau]\) are closed under various operations.

**Lemma 4.5.** Let \(M\) be a CBPV model. For each \(\tau\), 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
\[
g^v \subseteq f^v \land f^v \mathcal{R}[\tau]_W f^n \land f^n \subseteq g^n \implies g^v \mathcal{R}[\tau]_W g^n
\]
2. For all \(w: W' \rightarrow W\), and \(f^v, f^n\), we have
\[
f^v \mathcal{R}[\tau]_W f^n \implies (f^v \circ w) \mathcal{R}[\tau]_{W'} (f^n \circ w)
\]
3. For all \(f^v, f^n\), we have
\[
f^v \mathcal{R}[\tau]_{W \times X} f^n \implies (f^v)^{W \times X} \mathcal{R}[\tau]_{W \times T_X} (f^n)^{W \times X}
\]
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
\[
f_1^v \mathcal{R}[\tau]_{W_1} f_1^n \land f_2^v \mathcal{R}[\tau]_{W_2} f_2^n \implies [f_1^v, f_2^v] \mathcal{R}[\tau]_{W_1 + W_2} [f_1^n, f_2^n]
\]

**Proof.** The proof of each property is by induction on the type \(\tau\).

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 \( g^n, g^u \), that \( g^n \mathcal{R}[\tau]_{W''} g^u \) implies

\[
(\text{ev}^\uparrow \circ \langle\langle f^u \circ w \circ w', g^u \rangle\rangle) \mathcal{R}[\tau'] (\text{ev} \circ (f^n \circ w \circ w', g^n))
\]

This follows immediately from the assumption \( f^u \mathcal{R}[\tau \to \tau']_W f^n, \) 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 types follows from the inductive hypothesis, by naturality of extension operators and the definition of the product types is immediate from the inductive hypothesis. For a function type \( \tau \to \tau' \) we show, for every \( w : W' \to W \times TX \) and \( g^n, g^u \) satisfying \( g^n \mathcal{R}[\tau]_{W}, g^u \), that

\[
(\text{ev}^\uparrow \circ \langle\langle f^u \rangle\rangle \circ w, g^u) \mathcal{R}[\tau']_{W'} (\text{ev} \circ (f^n \circ \pi_1, g^n) \circ w, g^u)
\]

By property (2) above, we have

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

so that \( f^u \mathcal{R}[\tau \to \tau']_{(W \times X) \times W'} \) \( f^n \) implies

\[
(\text{ev}^\uparrow \circ \langle\langle f^u \circ \pi_1, g^u \circ \pi_2 \rangle\rangle) \mathcal{R}[\tau']_{(W \times X) \times W'} (\text{ev} \circ (f^n \circ \pi_1, g^n \circ \pi_2))
\]

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

\[
h^u = (\text{ev}^\uparrow \circ \langle\langle f^u \circ \pi_1, g^u \circ \pi_2 \rangle\rangle \circ \beta_{W,W',X}) (W \times X \to W') \circ \beta_{W,TX,W'} \circ \langle w, id_{W'} \rangle
\]

\[
h^n = (\text{ev} \circ (f^n \circ \pi_1, g^n \circ \pi_2) \circ \beta_{W,W',X}) (W \times X \to W') \circ \beta_{W,TX,W'} \circ \langle w, id_{W'} \rangle
\]

with \( \beta \) as in Definition 3.5. It then remains to show that \( h^u \) 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.

\[
h^u = \text{ev}^\uparrow \circ \langle\langle f^u \circ \pi_1, g^u \circ \pi_2 \rangle\rangle \circ \beta_{W,W',X} (W \times X \to W') \circ \beta_{W,TX,W'} \circ \langle w, id_{W'} \rangle
\]

\[
= \text{ev}^\uparrow \circ \langle\langle f^u \rangle\rangle \circ (f^u \times id) \circ \beta_{W,T(V[\tau]),X} (W \times X \to W') \circ \beta_{W,TX,T(V[\tau])} \circ \langle w, g^u \rangle
\]

\[
= \text{ev}^\uparrow \circ \langle\langle f^u \rangle\rangle \times \text{id}_T (V[V[\tau]]) \circ \langle w, g^u \rangle
\]

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

\[
h^n = (\text{ev} \circ (f^n \times id_{U_T(V[\tau])}) \circ \beta_{W,U_T(V[\tau]),X}) (W \times X \to W') \circ \beta_{W,TX,U_T(V[\tau])} \circ \langle w, g^n \rangle
\]

\[
= \text{ev} \circ (f^n \times id_{U_T(V[\tau])}) \circ \langle w, g^n \rangle
\]

\[
= \text{ev} \circ (f^n \times \text{id}_{U_T(V[\tau])}) \circ \langle w, g^n \rangle
\]
Lemma 4.6. The unit and bool cases are immediate from monotonicity of the copairing operator \([-,-]\). For product types, it is enough to note that \(T\pi_i \circ [f_1^\tau, f_2^\tau] = [T\pi_i \circ f_1^\tau, T\pi_i \circ f_2^\tau]\) and \(\pi_i \circ [f_1^\tau, f_2^\tau] = [\pi_i \circ f_1^\tau, \pi_i \circ f_2^\tau]\), and then apply the inductive hypothesis. For a function type \(\tau \to \tau'\), we show that \(g^\tau \mathcal{R}[\tau]_W \ g^n \) implies
\[ (ev^\tau \circ \langle\langle f_1^\tau, f_2^\tau \rangle \circ w, g^n \rangle) \mathcal{R}[\tau']_W \ (ev \circ \langle f_1^n, f_2^n \rangle \circ w, g^n) \]
where \(w : W' \to W_1 + W_2\). To do this, consider the following objects \(W'_1, W'_2\) and morphisms \(w'_1, w'_2\).
\[ W'_i = W' \times W_i \quad w'_i = \pi_2 : W'_i \to W_i \quad (i \in \{1,2\}) \]
Property (2) implies \((g^\tau \circ \pi_1) \mathcal{R}[\tau]_W (g^n \circ \pi_1)\), so from the assumption \(f_i^\tau \mathcal{R}[\tau \to \tau']_W f_i^n\), we obtain \(k_i^\tau \mathcal{R}[\tau']_W k_i^n\), where
\[ k_i^\tau = ev^\tau \circ \langle\langle f_i^\tau \circ w'_i, g^\tau \circ \pi_1 \rangle \quad k_i^n = ev \circ \langle f_i^n \circ w'_i, g^n \circ \pi_1 \rangle \]
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 \(\tau\), and then (2), to obtain
\[ ([k_1^\tau, k_2^\tau] \circ h) \mathcal{R}[\tau']_W ([k_1^n, k_2^n] \circ h) \]
where
\[ h : W' \xrightarrow{(id_{W'}, w)} W' \times (W_1 + W_2) \xrightarrow{\cong} W'_1 + W'_2 \]
The result follows because
\[ [k_1^\tau, k_2^\tau] \circ h = ev^\tau \circ \langle\langle f_1^\tau, f_2^\tau \rangle \circ w, g^\tau \rangle \quad [k_1^n, k_2^n] \circ h = ev \circ \langle f_1^n, f_2^n \rangle \circ w, g^n \]

The second lemma consists of some technical consequences of lax discardability, lax copyability and lax centrality; we state them here for use in the proof of Theorem 4.7 below. For convenience, we render each of the inequalities in the statement of the lemma in the syntax of CBPV (we will not need the syntactic inequalities in the following, so we omit the precise statements and proof).

**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
\[ T\pi_i \circ \langle\langle f_1, f_2 \rangle \rangle \subseteq f_i : W \to TX_i \]
\[ M_1 \texttt{to} x_1. M_2 \texttt{to} x_2. \texttt{return} x_i \preceq_{\text{ctx}} M_i \quad (f_k = \lfloor M_k \rfloor) \]
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
\[ (g^{\mathbf{1} \times TX_2})^{\mathbf{1} \times TX_2} \circ (f_1 \times f_2) \subseteq (g^{\mathbf{1} \times TX_2})^{\mathbf{1} \times TX_2} \circ (f_1 \times f_2) : W_1 \times W_2 \to TY \]
\[ M_1 \texttt{to} x_1. M_2 \texttt{to} x_2. N \preceq_{\text{ctx}} M_2 \texttt{to} x_2. M_1 \texttt{to} x_1. N \quad (f_i = \lfloor M_i \rfloor, g = \lfloor N \rfloor) \]
If \(f_2\) is lax central, then
\[ (g^{\mathbf{1} \times TX_2})^{\mathbf{1} \times TX_2} \circ (f_1 \times f_2) \subseteq (g^{\mathbf{1} \times TX_2})^{\mathbf{1} \times TX_2} \circ (f_1 \times f_2) : W_1 \times W_2 \to TY \]
\[ M_2 \texttt{to} x_2. M_1 \texttt{to} x_1. N \preceq_{\text{ctx}} M_1 \texttt{to} x_1. M_2 \texttt{to} x_2. N \quad (f_i = \lfloor M_i \rfloor, g = \lfloor N \rfloor) \]
(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

\[
(h^{\times X} \circ (id_X, g))^\dagger \circ f \sqsubseteq (h^{\times Y} \circ (f \times id_Y))^\dagger \circ (id_W, g^\dagger \circ f) : W \to TZ
\]

\[
M \tau x, N \tau y, N' \simeq_{ctx} (M \tau x, N) \tau y, M \tau x, N' \quad (f = [M], g = [N], h = [N'])
\]

(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

\[
\langle \langle g_1, g_2 \rangle \rangle^\dagger \circ f \sqsubseteq \langle \langle g_1^\dagger \circ f, g_2^\dagger \circ f \rangle \rangle : W \to T(Y_1 \times Y_2)
\]

\[
M \tau x, N_1 \tau y_1, N_2 \tau y_2 \simeq_{ctx} (M \tau x, N_1) \tau y_1, (M \tau x, N_2) \tau y_2 \quad (f = [M], g_i = [N_i], h = [N'])
\]

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

\[
T\pi_1 \circ \langle \langle f_1, f_2 \rangle \rangle = T\pi_1 \circ T(id_{X_1} \times \langle \rangle_{X_2}) \circ \langle \langle f_1, f_2 \rangle \rangle
\]

\[
= T\pi_1 \circ \langle \langle f_1, T\langle \rangle_{X_2} \circ f_2 \rangle \rangle
\]

\[
\sqsubseteq T\pi_1 \circ \langle \langle f_1, \eta_1 \circ \langle \rangle_W \rangle \rangle
\]

\[
= T\pi_1 \circ (\eta_{X_1 \times 1})^{\Pi X_2} \circ \langle \langle f_1, \langle \rangle_W \rangle \rangle
\]

\[
= f_1
\]

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

\[
(g^{\times X_1} \circ \Pi X_2) \circ (f_1 \times f_2)
\]

\[
= g^\dagger \circ seq^{X_1, X_2} \circ (f_1 \times id_{TX_2}) \circ (id_{W_1} \times f_2)
\]

\[
\sqsubseteq g^\dagger \circ seq^{R X_1, X_2} \circ (f_1 \times id_{TX_2}) \circ (id_{W_1} \times f_2)
\]

\[
= (g^{\times X_2} \circ \Pi X_1 \times \square) \circ (f_1 \times f_2)
\]

For the other inequality, precomposing with the isomorphisms \( (\pi_2, \pi_1) \) swaps the roles of \( f_1 \) and \( f_2 \), so that we can reuse the first inequality.
(3) The following proves the result.
\[(h^{i X \times Y} \circ (i d_X, g)) \uparrow \circ f\]

\[= (h^{i X \times Y} \circ (i d_X \times g)) \uparrow \circ T(i d_X, i d_X) \circ f\]  
(naturality of extension)

\[\sqsubseteq (h^{i X \times Y} \circ (i d_X \times g)) \uparrow \circ \text{seq}^R_{X,Y} \circ \langle f, f \rangle\]  
(lax copyability)

\[= (((h^{i X \times Y} \circ (i d_X \times g)) \uparrow) \circ (f \times g)) \circ \langle i d_W, f \rangle\]  
(associativity, left unit)

\[\sqsubseteq ((h^{i X \times Y} \circ (f \times id_Y)) \uparrow) \circ \langle i d_W, f \rangle\]  
(associativity law)

\[= (h^{i X \times Y} \circ (f \times id_Y)) \uparrow \circ \langle i d_W, g \uparrow \circ f \rangle\]  
(naturality of extension)

\[\sqsubseteq (h^{i X \times Y} \circ (f \times id_Y)) \uparrow \circ \langle i d_W, g \uparrow \circ f \rangle\]  
(Lemma 4.6(2))

\[\sqsubseteq (h^{\uparrow X \times Y} \circ (f \times id_Y)) \uparrow \circ \langle i d_W, f \rangle\]  
(naturality of extension)

\[= (h^{\uparrow X \times Y} \circ (f \times id_Y)) \uparrow \circ \langle i d_W, f \rangle\]  
(naturality of extension)

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

\[\text{seq}^R_{Y_2, Y_1} \circ (g_2, g_1)) \uparrow \circ f \sqsubseteq \text{seq}^R_{Y_2, Y_1} \circ (g_2 \uparrow \circ f, g_1 \uparrow \circ f)\]

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

\[\text{seq}^R_{Y_2, Y_1} \circ (g_2, g_1)) \uparrow \circ f\]

\[= ((\eta^{\uparrow X \times Y_1} \circ (g_2 \times id_{Y_1})) \uparrow \circ \langle i d_X, g_1 \rangle) \uparrow \circ f\]  
(naturality of extension)

\[\sqsubseteq ((\eta^{\uparrow X \times Y_1} \circ (g_2 \times id_{Y_1})) \uparrow \circ (f \times id_{Y_1})) \circ \langle i d_W, g_1 \uparrow \circ f \rangle\]  
(Lemma 4.6(3))

\[= (\eta^{\uparrow X \times Y_1} \circ (g_2 \uparrow \circ f \times id_{Y_1}) \circ \langle i d_W, g_1 \uparrow \circ f \rangle\]  
(associativity law)

\[= \text{seq}^R_{Y_2, Y_1} \circ (g_2 \uparrow \circ f, g_1 \uparrow \circ f)\]  
(naturality of extension)

\[= \text{seq}^R_{Y_2, Y_1} \circ (g_2 \uparrow \circ f, g_1 \uparrow \circ f)\]

\[\square\]

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

**Theorem 4.7.** Let \(\mathcal{M} = (\mathcal{C}, T)\) 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

\[\mathcal{V}[e] \times \mathcal{N}[e]\]

**Proof.** Explicitly, we are required to show, for each expression \(e\), that for all families of morphisms

\[(f^e_i : W \to T(V[\tau_i]))_i \quad (f^n_i : W \to U\tau(\mathcal{N}[\tau_i]))_i\]

we have

\[f^e_i \mathcal{R}[\tau_i]_W \quad f^n_i \mathcal{R}[\tau_k]_W \quad f^a_k \mathcal{R}[\tau_l]_W \quad \Rightarrow \quad (\mathcal{V}[e])^\uparrow \circ \langle f^e_i \rangle \circ \langle f^n_i \rangle \circ (\mathcal{R}[\tau])_W \circ (\mathcal{N}[e] \circ (f^a_i))_i\]

We prove this by induction on \(e\).
For a variable \( x_j \), we are required to prove
\[
(T\pi_j \circ \langle f_i^n \rangle_i) \mathcal{R}[\tau_j] W f_i^n
\]
A simple induction on \( j \), using Lemma 4.6(1), tells us that
\[
T\pi_j \circ \langle f_i^n \rangle_i \sqsubseteq f_i^n
\]
The result then follows from the assumption \( f_i^n \mathcal{R}[\tau_j] W f_j^n \) via Lemma 4.5(1).

- For the expression \( () \), we are required to show
\[
\mathcal{R}[\mathbf{unit}] W (\eta_1 \circ ()_W)
\]
By definition of \( \mathcal{R}[\mathbf{unit}] \), this is the same as
\[
\mathcal{R}[\mathbf{unit}] W (\eta_1 \circ ()_W)
\]
which is immediate from lax discardability.

- For a pair \( (e_1, e_2) \), the inductive hypothesis tells us that
\[
\langle \mathcal{V}[e_j] \rangle^\dagger \circ \langle f_i^n \rangle_i \mathcal{R}[\tau_j] W (\mathcal{N}[e_j] \circ \langle f_i^n \rangle_i)
\]
for each \( j \in \{1, 2\} \). We also have
\[
\sqsubseteq T\pi_j \circ \langle \mathcal{V}[e_1], \mathcal{V}[e_2] \rangle^\dagger \circ \langle f_i^n \rangle_i
\]
(\text{Lemma 4.6(4)})
\[
\sqsubseteq \mathcal{V}[e_j] \circ \langle f_i^n \rangle_i
\]
(\text{Lemma 4.6(1)})
so that Lemma 4.5 implies
\[
\langle \mathcal{V}[e_1], \mathcal{V}[e_2] \rangle^\dagger \circ \langle f_i^n \rangle_i \mathcal{R}[\tau_j] W (\pi_j \circ \langle \mathcal{N}[e_1], \mathcal{N}[e_2] \rangle \circ \langle f_i^n \rangle_i)
\]
Hence
\[
\langle \mathcal{V}[e_1], \mathcal{V}[e_2] \rangle \circ \langle f_i^n \rangle_i \mathcal{R}[\tau_1 \times \tau_2] W (\langle \mathcal{N}[e_1], \mathcal{N}[e_2] \rangle \circ \langle f_i^n \rangle_i)
\]
as required.

- For \( \mathbf{fst} e \), we need to show
\[
(T\pi_1 \circ \langle \mathcal{V}[e] \rangle^\dagger \circ \langle f_i^n \rangle_i) \mathcal{R}[\tau_1^\dagger] W (\pi_1 \circ \langle \mathcal{N}[e] \circ \langle f_i^n \rangle_i)
\]
which is immediate from the inductive hypothesis and the definition of \( \mathcal{R}[\tau_1^\dagger \times \tau_2^\dagger] \).

- The \( \mathbf{snd} \) case is similar to the \( \mathbf{fst} \) case.

- For the expression \( \mathbf{true} \), we need to show
\[
(T(\mathbf{inl} \circ ()_{\mathcal{V}[\Gamma]}) \circ \langle f_i^n \rangle_i) \mathcal{R}[\mathbf{bool}] W (\eta_2 \circ \mathbf{inl} \circ ()_W)
\]
which means
\[
(T(\mathbf{inl} \circ ()_{\mathcal{V}[\Gamma]}) \circ \langle f_i^n \rangle_i) \sqsubseteq (\eta_2 \circ \mathbf{inl} \circ ()_W)
\]
This follows from lax discardability, and naturality of \( \eta \).

- The \( \mathbf{false} \) case is similar to the \( \mathbf{true} \) case.

- For \( \mathbf{if} \ e_0 \ \mathbf{then} \ e_1 \ \mathbf{else} \ e_2 \), the inductive hypothesis gives us
\[
g_0^v \sqsubseteq g_0^v \mathcal{R}[\tau] W g_1^v \quad g_2^v \mathcal{R}[\tau] W g_2^v
\]
where we define
\[
g_j^v = \mathcal{V}[e_j] \circ \langle f_i^n \rangle_i \quad g_j^v = \mathcal{N}[e_j] \circ \langle f_i^n \rangle_i \quad (j \in \{0, 1, 2\})
\]
By applying all of the closure properties of Lemma 4.5, we therefore have
\[
((g_1^v, g_2^v) \circ \text{dist})^{W \times \Box} \circ (id_W \times g_0^v) \circ (\text{id}, id)) \mathcal{R}[\tau] W ((g_1^v, g_2^v) \circ \text{dist})^{W \times \Box} \circ (id_W \times g_0^v) \circ (\text{id}, id))
\]
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

\[(|V[e_1], V[e_2]| \circ \text{dist})^{\square \times 2} = |V[e_1], V[e_2]|) \circ \text{dist}\]

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

\[\mathcal{V}[\text{if } e_0 \text{ then } e_1 \text{ else } e_2]^\dagger \circ \langle f_i^\tau \rangle_i\]

\[= \langle \langle [V[e_1], V[e_2]| \circ \text{dist})^{\square \times 2} \circ (id, V[e_0]) \rangle \circ \langle f_i^\tau \rangle_i\]

\[= \langle \langle [V[e_1], V[e_2]| \circ (f_i^\tau) \times id_2) \rangle \circ id_W, g_0^n \rangle \quad \text{(Lemma 4.6(3))}\]

\[= \langle (g_1^n, g_2^n) \circ \text{dist})^{\square \times 2} \circ (id_W \times g_0^n) \circ (id, id)\]

We also have

\[\mathcal{N}[\text{if } e_0 \text{ then } e_1 \text{ else } e_2 \circ (f_i^\tau) \rangle_i = \langle (g_1^n, g_2^n) \circ \text{dist})^{\square \times 2} \circ (id_W \times g_0^n) \circ (id, id)\]

so that Lemma 4.5(1) implies

\[\mathcal{V}[\text{if } e_0 \text{ then } e_1 \text{ else } e_2]^\dagger \circ \langle f_i^\tau \rangle_i \mathcal{R}[\tau']_W (\mathcal{N}[\text{if } e_0 \text{ then } e_1 \text{ else } e_2] \circ \langle f_i^\tau \rangle_i)\]

as required.

For a \(\lambda\)-abstraction \(\lambda x: \tau. e\) of type \(\tau \to \tau'\), consider arbitrary \(w: W' \to W\) and \(g^n, g^n\) such that

\[g^n \in \mathcal{R}[\tau'_W, \tau_W]\]

We need to show

\[(\text{ev}^\dagger \circ \langle \langle [V[\lambda x: \tau. e]^\dagger \circ \langle f_i^\tau \rangle_i \circ w), g^n \rangle \rangle) \mathcal{R}[\tau'_W, (\text{ev} \circ \langle \mathcal{N}[\lambda x: \tau. e] \circ \langle f_i^\tau \rangle_i \circ w, g^n \rangle)\]

Applying Lemma 4.5(3) to the assumptions yields

\[(f_i^\tau \circ w) \mathcal{R}[\tau'_W, (f_i^n \circ w)\]

so that, by the inductive hypothesis, we have

\[(V[e]^\dagger \circ \langle \langle [f_i^\tau \circ w \rangle_i, g^n \rangle) \mathcal{R}[\tau'_W, (\mathcal{N}[e] \circ \langle (f_i^n \circ w) \rangle_i, g^n \rangle)\]

We show that this is the required instance of \(\mathcal{R}[\tau'_W, \tau_W]\) by rewriting both sides as follows. For the left-hand side, we have

\[\text{ev}^\dagger \circ \langle \langle [V[\lambda x: \tau. e]^\dagger \circ \langle f_i^\tau \rangle_i \circ w), g^n \rangle \rangle\]

\[= \text{ev}^\dagger \circ \langle T(\Lambda(V[e])) \circ \langle f_i^\tau \circ w \rangle_i, g^n \rangle\]

\[= (\text{ev} \circ \langle \Lambda(V[e]) \times id_{\mathcal{V}[\tau'_W]} \rangle)^\dagger \circ \langle \langle [f_i^\tau \circ w \rangle_i, g^n \rangle \quad \text{(naturality of extension)}\]

\[= \mathcal{V}[e]^\dagger \circ \langle \langle [f_i^\tau \circ w \rangle_i, g^n \rangle\]

For the right-hand side:

\[\text{ev} \circ \langle \mathcal{N}[\lambda x: \tau. e] \circ \langle f_i^n \rangle_i \circ w, g^n \rangle = \text{ev} \circ \langle \Lambda(\mathcal{N}[e]) \circ \langle f_i^n \circ w \rangle_i, g^n \rangle\]

\[= \mathcal{N}[e] \circ \langle (f_i^n \circ w) \rangle_i, g^n \rangle\]
• For an application \( ee' \), where \( e \) has type \( \tau \to \tau' \), the inductive hypothesis for \( e' \) gives us

\[
(\mathcal{V}[e'] \uparrow \circ \langle f^n_i \rangle_i) \mathcal{R}[\tau]_W (\mathcal{N}[e'] \circ \langle f^n_i \rangle_i)
\]

so that, by the inductive hypothesis for \( e \) with \( w = id_W \), we have

\[
(\mathcal{V}[e'] \uparrow \circ \langle f^n_i \rangle_i, (\mathcal{V}[e'] \uparrow \circ \langle f^n_i \rangle_i)) \mathcal{R}[\tau]_W (ev \circ \langle (\mathcal{N}[e] \circ \langle f^n_i \rangle_i), (\mathcal{N}[e'] \circ \langle f^n_i \rangle_i)\rangle)
\]

We rewrite both sides as follows. For the left we have

\[
\mathcal{V}[e e'] \uparrow \circ \langle f^n_i \rangle_i = (ev^{\mathcal{V}[\tau' \to \tau] \times (\mathcal{V}[\tau]_W)}) \circ \langle \mathcal{V}[e], \mathcal{V}[e'] \rangle \circ \langle f^n_i \rangle_i \quad \text{(naturalness of extension)}
\]

\[
= ev \uparrow \circ \langle \mathcal{V}[e], \mathcal{V}[e'] \rangle \uparrow \circ \langle f^n_i \rangle_i \quad \text{(left unit, associativity)}
\]

\[
= ev \uparrow \circ \langle (\mathcal{V}[e'] \uparrow \circ \langle f^n_i \rangle), (\mathcal{V}[e'] \uparrow \circ \langle f^n_i \rangle) \rangle \quad \text{(Lemma 4.6(4))}
\]

and for the right,

\[
\mathcal{N}[e e'] \circ \langle f^n_i \rangle_i = ev \circ \langle (\mathcal{N}[e] \circ \langle f^n_i \rangle_i), (\mathcal{N}[e'] \circ \langle f^n_i \rangle_i)\rangle
\]

We therefore have

\[
(\mathcal{V}[e e'] \uparrow \circ \langle f^n_i \rangle_i) \mathcal{R}[\tau']_W (\mathcal{N}[e e'] \circ \langle f^n_i \rangle_i)
\]

as required. \( \square \)

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

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

\[
\mathcal{V}[e] \preceq \mathcal{N}[e]
\]

**Proof.** By Theorem 4.7 we have \( \mathcal{V}[e] \preceq \mathcal{N}[e] \), so in particular \( \mathcal{V}[e] \mathcal{R}[\text{bool}]_I \mathcal{N}[e] \) By definition, the latter means \( \mathcal{V}[e] \subseteq \mathcal{N}[e] \), which implies the result by adequacy. \( \square \)

The conclusion of this corollary, namely \( \mathcal{V}[e] \preceq \mathcal{N}[e] \) is independent of the choice of model \( \mathcal{M} \). In contrast, the conclusion of Theorem 4.7 is not independent of \( \mathcal{M} \). Theorem 4.7 should therefore be viewed as a result about the denotations \( \mathcal{V}[e] \) and \( \mathcal{N}[e] \), rather than about the translations \( \mathcal{V}[e] \) and \( \mathcal{N}[e] \). We rectify this in Section 6 below, where the conclusion of our main result Theorem 6.2 relates \( \mathcal{V}[e] \) with \( \mathcal{N}[e] \) in terms of the contextual preorder, which is independent of \( \mathcal{M} \).

### 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 \( \preceq \) for each of these examples, by using the fact that, in each of the three \( \text{Poset} \)-categories \( \text{Set}, \text{Poset}, \omega \text{Cpo} \), 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.
\[ t^v \mathcal{R}'[\tau] t^n \text{ (where } t^v \in T(\mathcal{V}[\tau]) \text{ and } t^n \in U_1(\mathcal{N}[\tau])) \]

\[ t^v \mathcal{R}'[\text{unit}] t^n \iff t^v \sqsubseteq t^n \]

\[ t^v \mathcal{R}'[\tau_1 \times \tau_2] t^n \iff t_{\pi_1} t^v \mathcal{R}'[\tau_1] \pi_1 t^n \land T \pi_2 t^v \mathcal{R}'[\tau_2] \pi_2 t^n \]

\[ t^v \mathcal{R}'[\text{bool}] t^n \iff t^v \sqsubseteq t^n \]

\[ t^v \mathcal{R}'[\tau \to \tau'] t^n \iff \forall a^v, a^n. a^v \mathcal{R}'[\tau] a^n \Rightarrow ev^v(seq^L(t^v, a^n)) \mathcal{R}'[\tau'] t^n a^n \]

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

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

\[ f^v \mathcal{R}[\tau]_W f^n \iff \forall w \in W. f^v w \mathcal{R}'[\tau] f^n w \]

for every \( f^v : W \to T(\mathcal{V}[\tau]) \) and \( f^n : W \to U_1(\mathcal{N}[\tau]) \). Hence, for each \( g^v : \mathcal{V}[\Gamma] \to T(\mathcal{V}[\tau]) \) and \( g^n : \mathcal{N}[\Gamma] \to U_1(\mathcal{N}[\tau]) \), we have

\[ g^v \prec g^n \]

exactly when, for all \( (a^v_i \in T(\mathcal{V}[\tau_i]))_i \) and \( (a^n_i \in U_1(\mathcal{N}[\tau_i]))_i \),

\[ a^v_1 \mathcal{R}'[\tau_1] a^n_1 \land \cdots \land a^v_k \mathcal{R}'[\tau_k] a^n_k \Rightarrow (g^v \downarrow \langle a^v_i \rangle_i) \mathcal{R}'[\tau'] (g^n \downarrow \langle a^n_i \rangle_i) \]

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

**Proof.** By induction on \( \tau \). This is trivial for \( \text{unit, bool} \) and product types. For a function type \( \tau \to \tau' \), the \((\iff)\) 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 effects example, we have

\[ \mathcal{V}[e] = \mathcal{N}[e] \]

for every \( \Gamma \vdash e : \tau \), and in particular,

\[ \exists V : \text{bool. } (\mathcal{V}[e'] \downarrow \text{return } V) \land (\mathcal{N}[e'] \downarrow \text{return } V) \]

for every closed expression \( e' : \text{bool} \).

**Proof.** This is immediate from Theorem 4.7 and Corollary 4.8, with \( \preceq = \preceq_{\text{pure}} \).

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

\[ \mathcal{V}[e] = \mathcal{N}[e] \]

for every \( \Gamma \vdash e : \tau \), and in particular,

\[ \forall V : \text{bool. } (\mathcal{V}[e'] \downarrow \text{return } V) \Rightarrow (\mathcal{N}[e'] \downarrow \text{return } V) \]

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

\[
\begin{align*}
& h^v : \mathcal{V} [\tau \rightarrow \tau'] \times \mathcal{V} [\tau] \rightarrow T(\mathcal{V} [\tau']) \\
& h^n : U_T(\mathcal{N} [\tau \rightarrow \tau']) \times U_T(\mathcal{N} [\tau]) \rightarrow U_T(\mathcal{N} [\tau'])
\end{align*}
\]

are \( \omega \)-continuous functions that satisfy

\[
\begin{align*}
\tau & \rightarrow \tau' \\
\lambda x. e & \Rightarrow h^v(h^n(t^v, a^v), h^n(t^n, a^n))
\end{align*}
\]

when \( t^v, a^v \neq \perp \), then we have

\[
\text{fix}(\lambda g^v. h^v(g^v, -)) \Rightarrow \text{fix}(\lambda g^n. h^n(g^n, -))
\]

This follows from the fact that each \( R' \) relates \( \perp \) to \( \perp \) and is closed under least upper bounds of \( \omega \)-chains, which can be proved by a simple induction on \( \tau' \).

The second part follows from Corollary 4.8, by adequacy of the model with respect to the program relation defined in Example 2.4.

Corollary 4.12. For our nondeterminism example, we have

\[
\forall V : \mathcal{V} [e] \times \mathcal{N} [e]
\]

for every \( \Gamma \vdash e : \tau \), and in particular,

\[
\forall V : \mathcal{V} [e'] \downarrow \text{return } V \Rightarrow (\mathcal{N} [e'] \downarrow \text{return } V)
\]

for every closed expression \( e' : \mathcal{V} [e] \times \mathcal{N} [e] \).

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

\[
\perp \Rightarrow t^v_1 R' [\tau'] t^v_1 \land t^v_2 R' [\tau'] t^v_2 \Rightarrow (t^v_1 \sqcup t^v_2) R' [\tau'] (t^n_1 \sqcup t^n_2)
\]

This is a simple induction on \( \tau' \). 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

\[
\forall V : \mathcal{V} [e] \times \mathcal{N} [e]
\]

for every \( \Gamma \vdash e : \tau \), and in particular,

\[
\forall b \in \{ \text{true}, \text{false} \}. \exists V : \mathcal{V} [e] \downarrow \text{return } V \wedge (\mathcal{N} [e] \downarrow \text{return } V)
\]

for every closed expression \( e' : \mathcal{V} [e] \times \mathcal{N} [e] \).

Proof. Once again, we need to add the extra case to Theorem 4.7. It is enough to show that \( (\text{get} \circ (\downarrow \mathcal{V}[e]) \circ (\downarrow f)) \circ (\downarrow w) \) \( R' [\mathcal{V} [\mathcal{V} [\text{bool}]] \downarrow \text{get} \circ (\downarrow w) \), 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 effects), we prove a statement (Theorem 5.10) of the form

$$\mathcal{V}[e] \subseteq \psi_T \circ \mathcal{N}[e] \circ \hat{\phi}_T$$

Involving morphisms $\phi_T$ and $\psi_T$ for mapping between the call-by-value and call-by-name semantics:

$$(\mathcal{T}(\mathcal{V}[\tau]) \xleftarrow{\phi_T} \mathcal{U}(\mathcal{N}[\tau]))$$

(In the inequality above, $\hat{\phi}_T : \mathcal{V}[\Gamma] \to \mathcal{N}[\Gamma]$ is constructed by extending the morphisms $(\phi_T \circ \eta) : \mathcal{V}[\tau] \to \mathcal{U}(\mathcal{N}[\tau])$ 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 $\phi_T \circ (-)$ and $\psi_T \circ (-)$) form Galois connections [MSS86]. This is the crucial property that enables us to prove the inequality above.\(^\text{5}\)

**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 $x \subseteq \psi(\phi x)$ for all $x \in X$ and $\phi(\psi y) \subseteq y$ 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 $\phi_T$ and $\psi_T$ that represent the relations $\mathcal{R}[\tau]$, i.e. that satisfy the following equivalences.

$$\phi_T \circ f^v \subseteq f^n \iff f^v \mathcal{R}[\tau]W f^n \iff f^v \subseteq \psi_T \circ f^n$$

These equivalences uniquely determine $\psi_T$ and $\phi_T$, and guarantee that we have Galois connections $(\psi_T \circ (-), \phi_T \circ (-))$. Furthermore, these equivalences enable us to prove

$$\mathcal{V}[e] \subseteq \psi_T \circ \mathcal{N}[e] \circ \hat{\phi}_T$$

as a corollary of $\mathcal{V}[e] \times \mathcal{N}[e]$. 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 $\mathcal{M} = (\mathcal{C}, \mathcal{T})$, the morphisms $\phi_T$ and $\psi_T$ are defined in Figure 7.\(^\text{6}\) As for the relations $\mathcal{R}[\tau]$, the definition is by induction on $\tau$. This is mutual induction: at contravariant positions, the definition of $\phi$ uses $\psi$, 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 $(\phi_T \circ (-), \psi_T \circ (-))$, equivalently, that we have

$$\phi_T \circ \psi_T \subseteq id_{U_T(\mathcal{N}[\tau])} \quad id_{T(\mathcal{V}[\tau])} \subseteq \psi_T \circ \phi_T$$

\(^\text{5}\)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.

\(^\text{6}\)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].
The round-trip from call-by-value to call-by-name and back thunks the computational effects.

\[
\phi_{\text{unit}} = id_1 : T1 \rightarrow T1
\]

\[
\phi_{\text{unit}} \times \text{unit} = (\phi_{\text{unit}} \circ T \pi_1, \phi_{\pi_2} \circ T \pi_2) : T(\mathcal{V}[\tau_1] \times \mathcal{V}[\tau_2]) \rightarrow U_T(\mathcal{N}[\tau_1]) \times U_T(\mathcal{N}[\tau_2])
\]

\[
\phi_{\text{bool}} = id_2 : T2 \rightarrow T2
\]

\[
\phi_{\text{unit}} \times \text{unit} = (\phi_{\text{unit}} \circ ev \circ (\langle \pi_1, \psi_{\pi_2} \rangle)) : T(\mathcal{V}[\tau] \Rightarrow T(\mathcal{V}[\tau'])) \rightarrow U_T(\mathcal{N}[\tau]) \Rightarrow U_T(\mathcal{N}[\tau'])
\]

\[
\phi_r : T(\mathcal{V}[\tau]) \rightarrow U_T(\mathcal{N}[\tau])
\]

\[
\psi_{\text{unit}} = id_T1 : T1 \rightarrow T1
\]

\[
\psi_{\text{unit}} \times \text{unit} = (\psi_{\text{unit}} \circ \pi_1, \psi_{\pi_2} \circ \pi_2) : U_T(\mathcal{N}[\tau_1]) \times U_T(\mathcal{N}[\tau_2]) \rightarrow T(\mathcal{V}[\tau_1] \times \mathcal{V}[\tau_2])
\]

\[
\psi_{\text{bool}} = id_T2 : T2 \rightarrow T2
\]

\[
\psi_{\text{unit}} \times \text{unit} = (\psi_{\text{unit}} \circ ev \circ (id \times (\phi_r \circ \eta))) : U_T(\mathcal{N}[\tau]) \Rightarrow U_T(\mathcal{N}[\tau']) \Rightarrow T(\mathcal{V}[\tau] \Rightarrow T(\mathcal{V}[\tau']))
\]

\[
\psi_r : U_T(\mathcal{N}[\tau]) \rightarrow T(\mathcal{V}[\tau])
\]

Figure 7. Semantic morphisms \(\phi\) from call-by-value to call-by-name and \(\psi\) from call-by-name to call-by-value.

Now consider what happens when we convert a lazy pair \(N = \lambda\{1. N_1, 2. N_2\}\) of type \(N(\text{unit} \times \text{unit}) = \text{Funit} \times \text{Funit}\) into call-by-value, and then back into call-by-name:

\[
\left[\lambda\{1. N_1 \rightarrow z_1, 2. N_2 \rightarrow z_2, \text{return} z_1, \text{return} z_2\}\right] = (\phi_{\text{unit}} \times \text{unit} \circ \psi_{\text{unit}} \times \text{unit} \circ [N]) \subseteq [N]
\]

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

\[
[M] \subseteq (\psi_{\text{unit}} \times \text{unit} \circ \phi_{\text{unit}} \times \text{unit} \circ [M]) = \left[M \rightarrow z, \text{match} z \text{ with } (z_1, z_2) = \left[M \rightarrow \text{match} z' \text{ with } (z_1', z_2') = \text{return} (z_1, z_2)\right]\right]
\]

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 \(M : F(\mathcal{V}[\text{unit} \rightarrow \text{unit}]) = F(U(\text{unit} \rightarrow \text{Funit}))\) to call-by-name and then back to call-by-value. By doing this we obtain the denotation of a computation that immediately returns:

\[
[M] \subseteq (\psi_{\text{unit}} \rightarrow \text{unit} \circ \phi_{\text{unit}} \rightarrow \text{unit} \circ [M]) = \left[\text{return thunk } \lambda x : \text{unit}. M \rightarrow z, x \rightarrow \text{force} z\right]
\]

The round-trip from call-by-value to call-by-name and back thunks the computational 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 $C$. 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 $Y \in |C|$), then we say that $T$ is lax idempotent.$^7$

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 $\Gamma \vdash_c M : FA$ be a computation. For every adequate CBPV model, if $J M$ is lax thunkable, then $M \texttt{to } x. \texttt{return thunk return } x \equiv_{\text{ctx}}^\Gamma \texttt{return thunk } M$

Proof. We have

$$[M \texttt{to } x. \texttt{return thunk return } x] = T[\eta_A] \circ [M] \sqsubseteq \eta_{T[\eta_A]} \circ [M] = [\texttt{return thunk } M]$$

which implies the result by adequacy. \qed

Example 5.4. For three of our examples the strong Poset-monad $T$ is lax idempotent. For no 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 $\perp$ when $t = \perp$ (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

$$T\eta_Y S = \downarrow\{ y \} \subseteq S \quad (\text{intuitively, we can postpone nondeterministic choices}).$$

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 $f x \texttt{true} = f x \texttt{false}$ for all $x \in X$. In particular, $[\texttt{get}]$ is not lax thunkable. 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. Indeed, 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]. A different model, such as the identity monad on $\texttt{Set} \times \texttt{Set}$, which is lax idempotent, may enable us to apply Theorem 6.2.

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

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

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

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].
(3) the implication
\[ g_1 \sqsubseteq (g_2 \circ (\eta_Y \times id_W))^{\square \times W} \Rightarrow g_1 \circ (f \times id_W) \sqsubseteq g_2 \circ (f \times id_W) \]
holds for all \( T \)-algebras \( Z \) and morphisms \( g_1, g_2 : TY \times W \to Z \);

(4) the implication
\[ g_1 \sqsubseteq (g_2 \circ \eta_Y)^\dagger \Rightarrow g_1 \circ f \sqsubseteq g_2 \circ f \]
holds for all \( T \)-algebras \( Z \) and morphisms \( g_1, g_2 : TY \to Z \).

Proof. (1) \(\Rightarrow\) (2): If \( g_1 \sqsubseteq (g_2 \circ (id_W \times \eta_Y))^{\dagger W \times \square} \) then
\[
g_1 \circ (id_W \times f) \sqsubseteq (g_2 \circ (id_W \times \eta_Y))^\dagger W \times \square \circ (id_W \times f) \quad \text{(assumption)}
= g_2^\dagger W \times \square \circ (id_W \times T \eta_Y) \circ (id_W \times f) \quad \text{(naturality of extension)}
\sqsubseteq g_2^\dagger W \times \square \circ (id_W \times \eta_{TY}) \circ (id_W \times f) \quad \text{(lax thunkability of } f \text{)}
= g_2 \circ (id_W \times f) \quad \text{(left unit law)}
\]

(2) \(\Rightarrow\) (4): Specializing (2) to \( W = 1 \) yields (4).

(4) \(\Rightarrow\) (1): Consider the \( T \)-algebra \( Z = F_T(TY) \) 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) \(\Rightarrow\) (3): Similar to the proof that (1) implies (2).

(3) \(\Rightarrow\) (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
\[ T\langle id_Y, id_Y \rangle = (\eta_{Y \times Y} \circ \langle id_Y, id_Y \rangle)^\dagger = (seq_{Y,Y}^L \circ \langle \eta_Y, \eta_Y \rangle)^\dagger = ((seq_{Y,Y}^L \circ \langle id_Y, id_Y \rangle) \circ \eta_Y)^\dagger \]
so Lemma 5.5(4) implies
\[ T\langle id_Y, id_Y \rangle \circ f \sqsubseteq (seq_{Y,Y}^L \circ \langle id_Y, id_Y \rangle) \circ f = seq_{Y,Y}^L \circ \langle f, f \rangle \]
Lax centrality: We have

\[
\text{seq}^L_{Y,W} = (\eta_Y \times W)^{\bigtriangleup \times \Box} \quad \text{(definition of seq}^L)\]

\[
= ((\eta_Y \times W)^{\Box \times \Box} \circ (\eta_Y \times \text{id}_W))^{Y \times \Box} \quad \text{(left unit law)}
\]

\[
= ((\eta_Y \times W)^{\Box \times \Box} \circ (\eta_Y \times \text{id}_TW))^{\Box \times TW} \quad \text{(naturality of extension)}
\]

\[
= (\text{seq}^R_{Y,W} \circ (\eta_Y \times \text{id}_TW))^{\Box \times TW} \quad \text{(definition of seq}^R)\]

so Lemma 5.5(3) implies

\[
\text{seq}^L_{Y,W} \circ (f \times \text{id}_TW) \sqsubseteq \text{seq}^R_{Y,W} \circ (f \times \text{id}_TW) \quad \square
\]

Our aim is now to establish the relationship between \(R[J]^{\tau} \) and \((\phi_{\tau}, \psi_{\tau})\) outlined at the beginning of this section. For \(\tau = \text{unit}\) and \(\tau = \text{bool}\) this turns out to be easy. For product types, we use lax discardability 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

\[
f \sqsubseteq \langle g_1, g_2 \rangle \iff T\pi_1 \circ f \sqsubseteq g_1 \land T\pi_2 \circ f \sqsubseteq g_2
\]

(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

\[
f \sqsubseteq \eta_{X \Rightarrow Z} \circ \Lambda g \iff \text{ev}^{\bigtriangleup \times X} \circ (f \times \text{id}_X) \sqsubseteq g
\]

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

\[
T\pi_1 \circ f \sqsubseteq T\pi_1 \circ \langle g_1, g_2 \rangle \quad \text{(assumption)}
\]

\[
\sqsubseteq g_i \quad \text{(Lemma 4.6(1))}
\]

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

\[
f = T(\pi_1 \times \pi_2) \circ (\text{id, id}) \circ f
\]

\[
\sqsubseteq T(\pi_1 \times \pi_2) \circ \text{seq}^L \circ \langle f, f \rangle \quad \text{(lax copyability)}
\]

\[
= \langle T\pi_1 \circ f, T\pi_2 \circ f \rangle \quad \text{(naturality of extension)}
\]

\[
\sqsubseteq \langle g_1, g_2 \rangle \quad \text{(assumption)}
\]

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

\[
\text{ev}^{\bigtriangleup \times X} \circ (f \times \text{id}_X) \sqsubseteq \text{ev}^{\bigtriangleup \times X} \circ (\eta_{X \Rightarrow Z} \circ \Lambda g) \times \text{id}_X \quad \text{(assumption)}
\]

\[
= \text{ev} \circ (\Lambda g \times \text{id}_X) \quad \text{(left unit law)}
\]

\[
= g
\]

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

\[
\eta_{X \Rightarrow Z} \circ \Lambda (\text{ev}^{\bigtriangleup \times X}) \circ f = \eta_{X \Rightarrow Z} \circ \Lambda (\text{ev}^{\bigtriangleup \times X} \circ (f \times \text{id}_X))
\]

\[
\sqsubseteq \eta_{X \Rightarrow Z} \circ \Lambda g \quad \text{(assumption)}
\]
so it suffices to show \( f \sqsubseteq \eta_{X \Rightarrow Z} \circ \Lambda((ev^{\sqcap_\times X}) \circ f) \). Since \( f \) is lax thunkable, this is a consequence of Lemma 5.5(4) applied to the following.

\[
\begin{align*}
\text{id}_{T(X \Rightarrow Z)} & = (\eta_{X \Rightarrow Z} \circ \Lambda ev)^\dagger \\
& = (\eta_{X \Rightarrow Z} \circ \Lambda (ev^{\sqcap_\times X} \circ (\eta_{X \Rightarrow Z} \times \text{id}_X)))^\dagger \\
& = (\eta_{X \Rightarrow Z} \circ \Lambda (ev^{\sqcap_\times X} \circ \eta_{X \Rightarrow Z}))^\dagger
\end{align*}
\]

(right unit law)

We can now relate \( \mathcal{R}[\tau] \) to the morphisms \( \phi_\tau \) and \( \psi_\tau \), as follows.

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

\[
\begin{align*}
f^v : W \rightarrow T(\mathcal{V}[\tau]) & \quad f^n : W \rightarrow U_T(\mathcal{N}[\tau])
\end{align*}
\]

we have the following equivalences.

\[
\phi_\tau \circ f^v \sqsubseteq f^n \iff f^v \; \mathcal{R}[\tau]_W \; f^n \iff f^v \sqsubseteq \psi_\tau \circ f^n
\]

**Proof.** By induction on the type \( \tau \).

- 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.

\[
\forall i \in \{1, 2\}. \; \phi_{\tau_i} \circ T_{\pi_1} \circ f^v \sqsubseteq \pi_i \circ f^n \quad \forall i \in \{1, 2\}. \; T_{\pi_1} \circ f^v \sqsubseteq \psi_{\tau_i} \circ \pi_i \circ f^n
\]

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 \( \pi_1 \circ \phi_{\tau_1 \times \tau_2} \circ f^v \sqsubseteq \pi_1 \circ f^n \) holds for all \( i \in \{1, 2\} \). The required equivalence therefore follows from

\[
\pi_1 \circ \phi_{\tau_1 \times \tau_2} \circ f^v = T_{\pi_1} \circ \phi_{\tau_1} \circ f^v
\]

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 \( \tau \rightarrow \tau' \), we consider the two equivalences separately.

  For the equivalence on the left, we have

\[
\phi_{\tau \rightarrow \tau'} \circ f^v \sqsubseteq f^n \iff \Lambda^{-1}(\phi_{\tau \rightarrow \tau'} \circ f^v) \sqsubseteq \Lambda^{-1}f^n
\]

\[
\iff \phi_{\tau \rightarrow \tau'} \circ ev^\dagger \circ \langle f^v \circ \pi_1, \psi_\tau \circ \pi_2 \rangle \sqsubseteq ev \circ (f^n \times \text{id})
\]

\[
\iff (ev^\dagger \circ \langle f^v \circ \pi_1, \psi_\tau \circ \pi_2 \rangle) \; \mathcal{R}[\tau']_{W \times U_T(\mathcal{N}[\tau])} \; (ev \circ (f^n \times \text{id}))
\]

where the second step uses the definition of \( \phi_{\tau \rightarrow \tau'} \), and the final step uses the inductive hypothesis. Call the instance of \( \mathcal{R}[\tau']_{W \times U_T(\mathcal{N}[\tau])} \) above \((*)\). To show that \( f^v \; \mathcal{R}[\tau \rightarrow \tau'] \; f^n \) follows from \((*)\), consider arbitrary \( w : W' \rightarrow W \) 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 \(<w, g^n>\) to obtain

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

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

\[
(\langle ev^\dagger \circ ev \circ \psi_\tau \circ g^n \rangle) \; \mathcal{R}[\tau']_{W} \; (ev \circ (f^n \circ w, g^n))
\]
as required. Conversely, \( f^v \mathcal{R}[^\tau \rightarrow ^\tau'] W f^n \) implies (*) by taking

\[
W' = W \times U(\mathcal{N}[^\tau]) \quad w = \pi_1 : W' \rightarrow W \quad g^v = \psi_\tau \circ \pi_2 \quad g^n = \pi_2
\]

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

\[
f^v \sqsubseteq \psi_{\tau \rightarrow ^\tau} \circ f^n \iff ev^{[\square \times V[\tau]]} \circ (f^v \times id) \sqsubseteq \psi_{\tau} \circ ev \circ(f^n \times (\phi_\tau \circ \eta))
\]

\[
\iff (ev^{[\square \times V[\tau]]} \circ (f^v \times id)) \mathcal{R}[^\tau] W \times V[^\tau] (ev \circ (f^n \times (\phi_\tau \circ \eta)))
\]

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

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

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

\[
(ev^{[\square \times V[\tau]]} \circ (f^v \times id))^{[\square \times \square]} \circ \langle w, g^n \rangle = (ev^{[\square \times V[\tau]]})^{[\mathcal{T}(V[t \rightarrow t']) \times \square]} \circ \langle f^v \circ w, g^n \rangle \quad \text{(naturality of extension)}
\]

\[
= (ev^{[\mathcal{T}(t \rightarrow t')] \times \square})^{[\square \times \mathcal{T}(V[\tau])]} \circ \langle f^v \circ w, g^n \rangle \quad \text{(Lemma 4.6(2))}
\]

\[
= ev^\dagger \circ \langle f^v \circ w, g^n \rangle \quad \text{(left unit, associativity)}
\]

Since \( g^v \mathcal{R}[^\tau] W' g^n \), we also have

\[
(ev \circ (f^n \times (\phi_\tau \circ \eta)))^{[\square \times \square]} \circ \langle w, g^n \rangle \sqsubseteq (ev \circ (f^n \times \phi_\tau)) \circ \langle w, g^n \rangle \quad \text{(Lemma 5.5(2))}
\]

\[
= ev \circ (f^n \circ w, \phi_\tau \circ g^n)
\]

\[
\sqsubseteq ev \circ (f^n \circ w, g^n) \quad \text{(inductive hypothesis)}
\]

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

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

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

\[
W' = W \times V[^\tau] \quad w = \pi_1 \quad g^v = \eta \circ \pi_2 \quad g^n = \phi_\tau \circ \eta \circ \pi_2
\]

noting that the inductive hypothesis implies \( (\eta \circ \pi_2) \mathcal{R}_W (\phi_\tau \circ \eta \circ \pi_2) \). From this we obtain

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

which simplifies to (**). by the left unit law. \( \square \)

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 \( \mathcal{M} = (\mathcal{C}, T) \) be a CBPV model such that \( T \) is lax idempotent. The monotone functions

\[
\mathcal{C}(W, T(\mathcal{V}[^\tau])) \xrightarrow[\psi_\circ(-)]{\phi_\circ(-)} \mathcal{C}(W, U_T(\mathcal{N}[^\tau]))
\]
form a Galois connection for every source-language type \( \tau \) and object \( W \in |C| \).

**Proof.** Immediate from Lemma 5.8. \( \square \)

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

\[
\hat{\phi}_\Gamma = (\phi_{\tau_i} \circ \eta_{\mathcal{V}[\tau_i]} \circ \pi_{x_i})_i : \mathcal{V}[\Gamma] \to \mathcal{N}[\Gamma]
\]

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

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

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

**Proof.** The inequality we need to establish is equivalently

\[
\mathcal{V}[e] \subseteq \psi_\tau \circ \mathcal{N}[e] \circ (\phi_{\tau_i} \circ \eta_{\mathcal{V}[\tau_i]} \circ \pi_{x_i})_i
\]

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

\[
(\mathcal{V}[e] \uparrow \circ (\eta_{\mathcal{V}[\tau_i]} \circ \pi_{x_i})_i) \mathcal{R}[\tau] \mathcal{V}[\Gamma] (\mathcal{N}[e] \circ (\phi_{\tau_i} \circ \eta_{\mathcal{V}[\tau_i]} \circ \pi_{x_i})_i)
\]

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

This theorem has a partial converse, as follows.

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

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

\[
ev = \mathcal{V}[x ()] \subseteq \psi_\tau \circ \mathcal{N}[x ()] \circ \hat{\phi}_\tau \circ \phi_\tau = \psi_\tau \circ \phi_\tau \circ \ev
\]

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

\[
id_{T(1 \Rightarrow T(\mathcal{V}[\tau']))} \subseteq \psi_{\text{unit} \to \tau'} \circ \phi_{\text{unit} \to \tau'} = \eta_{1 \Rightarrow T(\mathcal{V}[\tau'])} \circ id_{1 \Rightarrow T(\mathcal{V}[\tau'])} \uparrow
\]

which implies

\[
id_{T(T(\mathcal{V}[\tau']))} \subseteq \eta_{T(\mathcal{V}[\tau'])} \circ id_{T(\mathcal{V}[\tau'])} \uparrow
\]

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

\[
T\eta_{\mathcal{V}[\tau']} = \id_{T(T(\mathcal{V}[\tau']))} \circ T\eta_{\mathcal{V}[\tau']} \subseteq \eta_{T(\mathcal{V}[\tau'])} \circ id_{T(\mathcal{V}[\tau'])} \uparrow \circ T\eta_{\mathcal{V}[\tau']} = \eta_{T(\mathcal{V}[\tau'])}
\]

as required. \( \square \)
\[
\Gamma \vdash_c M : F(\mathcal{V}(\tau)) \iff \Gamma \vdash_c \Phi_\tau M : \mathcal{N}(\tau)
\]

\[
\Phi_{\text{unit}} M = M
\]

\[
\Phi_{\tau_1 \times \tau_2} M = \lambda \{1. M \text{ to } x. \text{match } x \text{ with } (z_1, z_2). (\hat{\Phi}_{\tau_1} z_1),
\]

\[
2. M \text{ to } x. \text{match } x \text{ with } (z_1, z_2). (\hat{\Phi}_{\tau_2} z_2)\}
\]

\[
\Phi_{\text{bool}} M = M
\]

\[
\Phi_{\tau \rightarrow \tau'} M = \lambda x : U(\mathcal{N}(\tau)). M \text{ to } f. \Psi_{\tau}(\text{force } x) \text{ to } y. \Phi_{\tau'}(y \cdot \text{force } f)
\]

\[
\Gamma \vdash_c N : \mathcal{N}(\tau) \iff \Gamma \vdash_c \Psi_\tau N : F(\mathcal{V}(\tau))
\]

\[
\Psi_{\text{unit}} N = N
\]

\[
\Psi_{\tau_1 \times \tau_2} N = \Psi_{\tau_1}(1 \cdot N) \text{ to } z_1. \Psi_{\tau_2}(2 \cdot N) \text{ to } z_2. \text{return } (z_1, z_2)
\]

\[
\Psi_{\text{bool}} N = N
\]

\[
\Psi_{\tau \rightarrow \tau'} N = \text{return thunk } \lambda x : \mathcal{V}(\tau). \Psi_{\tau'}((\text{thunk } (\Phi_\tau(\text{return } x))) \cdot N)
\]

**Figure 8.** Syntactic maps \(\Phi\) from call-by-value to call-by-name and \(\Psi\) from call-by-name to call-by-value.

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 \([\text{get}] : 1 \rightarrow \mathcal{T}2\) is not lax thunkable, so we do not have \(\mathcal{V}[e] \subseteq \psi_\tau \circ \mathcal{N}[e] \circ \hat{\phi}_\Gamma\) 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 \(\mathcal{V}[e]\):

\[
\mathcal{V}[e] \preceq_{\text{ctx}} \Psi_{\tau}(\mathcal{N}[e][\hat{\Phi}_\Gamma])
\]

More precisely, we render \(\phi_\tau\) and \(\psi_\tau\) in the syntax as maps \(\Phi_\tau\) from call-by-value computations to call-by-name computations, and \(\Psi_\tau\) from call-by-name to call-by-value.\(^8\)

These are defined, again by induction on \(\tau\), 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

\[
\hat{\Phi}_{\Gamma} = x_1 \mapsto \text{thunk } (\Phi_{\tau_1}(\text{return } x_1)), \ldots, x_k \mapsto \text{thunk } (\Phi_{\tau_k}(\text{return } x_k))
\]

\(^8\)We define \(\Phi\) and \(\Psi\) directly as maps from computations to computations, but we could instead have defined computations

\[
x : U(F(\mathcal{V}(\tau))) \vdash_c \Phi': \mathcal{N}(\tau) \quad x : U(\mathcal{N}(\tau)) \vdash_c \Psi': F(\mathcal{V}(\tau))
\]

and then recovered \(\Phi\) and \(\Psi\) modulo \(\beta\eta\)-laws for thunks, by substitution. This definition is slightly less convenient to work with however.
for converting a call-by-value context into a call-by-name context. This has the following typing:

\[ \mathcal{N}(\Gamma) \vdash_c N : C \mapsto \mathcal{V}(\Gamma) \vdash_c \hat{\Phi}_\tau N : C \]

The maps \( \Phi, \Psi \) and \( \hat{\Phi} \) are syntactic renderings of \( \phi, \psi \) 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 \( \Gamma \vdash_c M : F(\mathcal{V}(\tau)) \);
2. \( [\Psi_\tau N] = \psi_\tau \circ [N] \) for all \( \Gamma \vdash_c N : \mathcal{N}(\tau) \);
3. \( [N[\hat{\Phi}_\tau]] = [N] \circ \hat{\phi}_\tau \) for all \( \Gamma \vdash_c N : C \).

**Proof.** (1) and (2) are proved by mutual induction on the type \( \tau \), 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 \( \mathcal{V}(\tau) \):

\[ \mathcal{V}(\Gamma) \vdash_c \Psi_\tau(\mathcal{N}(\tau)[\hat{\Phi}_\tau]) : F(\mathcal{V}(\tau)) \]

We can therefore compare \( \mathcal{V}(\tau) \) with \( \Psi_\tau(\mathcal{N}(\tau)[\hat{\Phi}_\tau]) \) directly. In particular, it makes sense to replace \( \mathcal{V}(\tau) \) with \( \Psi_\tau(\mathcal{N}(\tau)[\hat{\Phi}_\tau]) \) 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 \( \mathcal{V}(\tau) \) in this way changes the behaviour of a computation.

Recall that a program relation \( \preceq \) is a preorder on CBPV programs, and that each program relation induces a contextual preorder \( \preceq_{ctx} \). Given any program relation \( \preceq \), to show that the call-by-value and call-by-name translations of source-language expressions are related by \( \preceq_{ctx} \) 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 \( M = (C, T) \) be a CBPV model that is adequate with respect to a given a program relation \( \preceq \). If \( T \) is lax idempotent, then for every source-language expression \( \Gamma \vdash e \vdash \tau \) we have

\[ \mathcal{V}(\tau) \preceq_{ctx} \Psi_\tau(\mathcal{N}(\tau)[\hat{\Phi}_\tau]) \]

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

The generality of this theorem comes from two sources. First, we consider arbitrary program relations \( \preceq \). 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).

For our first three examples (no 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 \( \Phi_\tau \) and \( \Psi_\tau \) on terms form a Galois connection (with respect to \( \preceq_{ctx} \)). In particular, we have

\[ M \preceq_{ctx} \Psi_\tau(\Phi_\tau M) \quad \Phi_\tau(\Psi_\tau N) \preceq_{ctx} N \]
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 \( C \), let \( \Omega_C \) be the diverging computation \( \text{rec } x : U_C. \text{force } x \) (which has type \( C \)). Then if \( \tau = \text{bool} \to \text{bool} \) and \( M = \Omega_{\text{bool}}(\forall f. \text{return false}) \), we do not have \( M \succ_{\text{ctx}} \Psi_{\tau}(\Phi_{\tau} M) \), because for \( E = (\square \to f. \text{return false}) \) the computation \( E[M] \) diverges but \( E[\Psi_{\tau}(\Phi_{\tau} M)] \Downarrow \text{return false} \). In this case we have \( \Psi_{\tau}(\Phi_{\tau} M) \approx_{\text{ctx}} \text{return thunk } \lambda x : \text{bool}. \Omega_{\text{bool}}. \) For a counterexample to \( \Phi_{\tau}(\Psi_{\tau} N) \succ_{\text{ctx}} N \), let \( \tau = \text{bool} \to \text{bool} \) and \( N = \lambda x : U(\text{F bool}). \text{return true} \). Then for \( E' = ((\text{thunk } \Omega_{\text{F bool}}) \cdot \square) \), the computation \( E'[\Phi_{\tau}(\Psi_{\tau} N)] \) diverges but \( E'[N] \Downarrow \text{return true} \). Here we have \( \Phi_{\tau}(\Psi_{\tau} N) \approx_{\text{ctx}} \lambda x : U(\text{F bool}). \text{force } x \to y. \text{return true} \).

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 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 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 \( \Phi \) and \( \Psi \). 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, Kuc98, 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 computational 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 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 computational 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


Galois connecting call-by-value and call-by-name


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