

LINEAR USAGE OF STATE *

RASMUS EJLERS MØGELBERG ^a AND SAM STATON ^b

^a IT University of Copenhagen, Denmark
e-mail address: mogel@itu.dk

^b Radboud University Nijmegen, Netherlands
e-mail address: s.staton@cs.ru.nl

ABSTRACT. We investigate the phenomenon that *every monad is a linear state monad*. We do this by studying a fully-complete state-passing translation from an impure call-by-value language to a new linear type theory: the enriched call-by-value calculus. The results are not specific to store, but can be applied to any computational effect expressible using algebraic operations, even to effects that are not usually thought of as stateful. There is a bijective correspondence between generic effects in the source language and state access operations in the enriched call-by-value calculus.

From the perspective of categorical models, the enriched call-by-value calculus suggests a refinement of the traditional Kleisli models of effectful call-by-value languages. The new models can be understood as enriched adjunctions.

1. INTRODUCTION

1.1. Informal motivation. The state-passing translation transforms a stateful program into a pure function. As an illustration, consider the following ML program which uses a single fixed memory cell `l : int ref`.

```
- fun f x = let val y = !l in l := x ; y end ;  
val f = fn : int -> int
```

The state-passing translation transforms that program into the following pure function which takes the state as an argument and returns the updated state as a result.

2012 ACM CCS: [Theory of computation]: Semantics and reasoning—Program semantics—Denotational semantics / Categorical semantics; Logic—Linear logic / Type theory.

Key words and phrases: Linear type theory, monads, computational effects, categorical semantics, enriched category theory, state passing translation.

* This article expands on a paper presented at the Fourth International Conference on Algebra and Coalgebra in Computer Science (CALCO 2011).

^a Research supported by the Danish Agency for Science, Technology and Innovation.

^b Research supported by EPSRC Fellowship EP/E042414/1, ANR Projet CHOCO, the Isaac Newton Trust, and ERC Projects ECSYM and QCLS.

```

- fun f (x,s) = let val y = s val s' = x in (y,s') end ;
val f = fn : int * int -> int * int

```

The state-passing translation is straightforward if the program only uses a fixed, finite area of memory of type \underline{S} : an impure program of type $A \multimap B$ becomes a pure program of type $A \times \underline{S} \rightarrow B \times \underline{S}$.

To what extent does the state-passing translation apply to programs with other effects? In this article we develop the idea that, from a semantic perspective, all effects can be understood as state effects. Central to our treatment of state is the idea of linear usage: in general, computations cannot copy the state and save it for later, nor can they discard the state and insert a new one instead. In 1972, Strachey wrote [46]:

The state transformation produced by obeying a command is essentially irreversible and it is, by the nature of the computers we use, impossible to have more than one version of σ [the state] at any one time.

Historically, the importance of the linearity of state arose in the study of programs with private store. In this setting, the naive state-passing translation does not preserve contextual equivalence. For instance, the function ‘snapback’ takes a stateful computation $f: A \multimap B$ and returns the computation that executes f but then snaps back to the original state:

$$\text{snapback} \stackrel{\text{def}}{=} \lambda f: (A \times \underline{S} \rightarrow B \times \underline{S}). \lambda(a, s): A \times \underline{S}. \langle \pi_1(f(a, s)), s \rangle: (A \multimap B) \rightarrow (A \multimap B)$$

The snapback program does not arise as the state-passing translation of an impure program. No impure program could tamper with the private store in the way that the snapback program does. In other words, the state-passing translation is not fully complete. One can use this fact to show that contextual equivalence is not preserved by the state-passing translation. Sieber [44] insisted that every function be wrapped in snapback to obtain full abstraction for his simple denotational model.

O’Hearn and Reynolds [32] resolved these difficulties with private store by moving to a linear typing system. Linear usage of state can be expressed syntactically by considering a stateful computation of type $A \multimap B$ as a linear map of type $!A \otimes \underline{S} \multimap !B \otimes \underline{S}$. The type of states \underline{S} must be used linearly, but the argument type A and the return type B can be used arbitrarily. The snapback program violates these linear typing constraints.

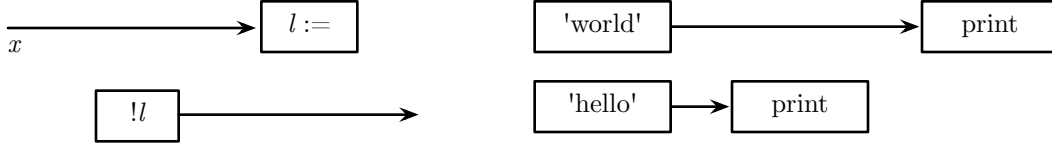
This notation is reminiscent of Girard’s linear logic [13]. Our starting point is actually a more refined calculus, the enriched effect calculus, which was developed by Egger, Møgelberg and Simpson [7, 8, 9, 10] as a way of investigating linear usage of resources such as state.

All effects are state effects. In this paper we develop this linear typing discipline to show that *all* effects can be understood as state effects and that there is *always* a fully-complete linear-use state-passing translation. Our analysis applies even to effects that do not involve the memory of a computer, like printing or coin-tossing. (For this reason we use the word ‘store’ to refer to memory related effects and ‘state’ for the general notion.) We now provide two informal explanations of this general phenomenon.

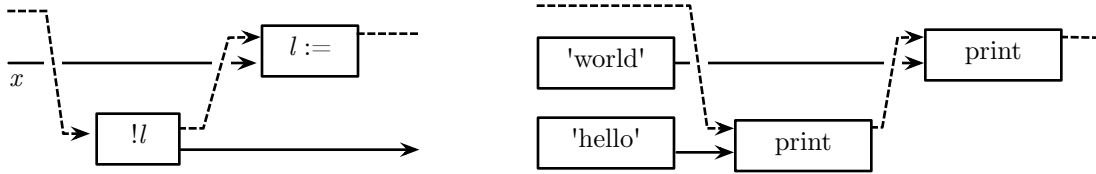
A first informal explanation is that an inhabitant of the ‘state’ type \underline{S} is an entire history of the universe. The history of the universe certainly cannot be discarded or duplicated. To be slightly more precise, if the effect in question is printing, then a ‘state’ is a string of everything that has been printed so far.

A second informal explanation involves Jeffrey’s graphical notation [19]. Jeffrey noticed that a naive graphical notation for composition of impure functions does not work, for it

describes how functions depend on their arguments but it does not describe the order of side effects:



To make the order of evaluation visible in the graphical notation, Jeffrey adds a special kind of edge which must be treated linearly. This is what we call state.



Our contribution in this paper is foundational, but let us speculate briefly on possible applications. State plays an important role in many aspects of semantics, including operational semantics and Hoare logic. The type $!A \otimes \underline{S}$ can be thought of as a type of configurations (program/state pairs) for a general abstract machine. This might pave the way for a general framework for operational semantics, building on the ideas of Plotkin and Power [35].

We also note that variations on linear-use state-passing style are already used to accommodate a broad class of effects within pure languages such as Clean [1] and Mercury [45].

1.2. The state-passing translation. The source language of our translation is an impure functional language with product and function types:

$$\sigma, \tau ::= 1 \mid \sigma \times \tau \mid \sigma \multimap \tau \mid \dots$$

We also consider sum types and unspecified base types. We adopt the call-by-value calling convention because it is most natural one for effectful programs. For simplicity we choose a fine-grain language in which the order of evaluation is totally explicit. The full syntax, equational theory and framework for denotational semantics is in Section 3. Sum types are treated in Section 6.

The types of the target language are essentially the minimal fragment of the enriched effect calculus (EEC, [8, 7]) that is needed for the linear-use state-passing translation. To enforce linear usage, EEC borrows some techniques and notation from linear logic ($!$, \otimes , and \multimap) but in a careful way, enforced by the following grammar:

$$\begin{aligned} A, B &::= 1 \mid A \times B \mid \underline{A} \multimap \underline{B} \mid \dots && \text{value types} \\ \underline{A}, \underline{B} &::= !A \otimes \underline{B} \mid \underline{S} \mid \dots && \text{computation types (underlined)} \end{aligned}$$

The full syntax, equational theory and framework for denotational semantics is in Section 2.

In Section 4 we provide a linear-use state passing translation from the source language to the target language. The translation on types takes a type τ of the source language to a value type $\tau^{\underline{S}}$ of the target language, by induction on the structure of types:

$$1^{\underline{S}} \stackrel{\text{def}}{=} 1 \quad (\sigma \times \tau)^{\underline{S}} \stackrel{\text{def}}{=} \sigma^{\underline{S}} \times \tau^{\underline{S}} \quad (\sigma \multimap \tau)^{\underline{S}} \stackrel{\text{def}}{=} !\sigma^{\underline{S}} \otimes \underline{S} \multimap !\tau^{\underline{S}} \otimes \underline{S} \quad \dots$$

Theorem 4.3 says that the linear-use state-passing translation is *fully complete*: it describes a bijective correspondence between programs in the source language of type τ and programs in the target language of type $\tau^{\underline{S}}$. This means that contextual equivalence is preserved by linear-use state-passing translation.

Enriched category theory. The constructions involved in the target language have an elegant categorical formulation: they can be modelled in any enriched category with copowers. For this reason we call the target language *the enriched call-by-value calculus*.

Recall that in categorical semantics a type A is interpreted as an object $\llbracket A \rrbracket$ of a category, a context Γ is interpreted as an object $\llbracket \Gamma \rrbracket$ of a category, and a term-in-context $\Gamma \vdash t : A$ is interpreted as a morphism $\llbracket t \rrbracket : \llbracket \Gamma \rrbracket \rightarrow \llbracket A \rrbracket$. In the enriched call-by-value calculus, there are two classes of type: computation types and value types. A categorical analysis must thus involve two categories: a category \mathbf{V} whose objects interpret value types, and a category \mathbf{C} whose objects interpret computation types. The structure of the types dictates the structure that the categories \mathbf{V} and \mathbf{C} must have:

- For the product types $(1, \times)$, \mathbf{V} must have finite products in the categorical sense.
- For the tensor type $(!A \otimes B)$, \mathbf{V} must act on \mathbf{C} . This means that there is a functor $(-)_1 \cdot (-)_2 : \mathbf{V} \times \mathbf{C} \rightarrow \mathbf{C}$ such that $1 \cdot \underline{X} \cong \underline{X}$ and $(A \times B) \cdot \underline{X} \cong A \cdot (B \cdot \underline{X})$.
- For the linear function type \multimap , the action $(-)_1 \cdot (-)_2$ must have a right adjoint in its first argument.

The linear function type \multimap ensures that the space of morphisms $\underline{X} \rightarrow \underline{Y}$ forms an object of \mathbf{V} . So \mathbf{C} is enriched in \mathbf{V} , and the action structure provides copowers. There are many examples of enriched models, including categories of algebras and Kleisli categories (understood as ‘closed Freyd categories’ [27]). Full details are in Section 2.

In Section 5 we explain that the connection with Kleisli categories gives us a categorical explanation of the linear-use state-passing translation. We use a semantic approach to prove the full completeness of the translation. We show that the traditional models of effectful call-by-value languages, using monads and Kleisli constructions, form a coreflective subcategory of the enriched models, and that the state-passing translation is the unit of the coreflection.

1.3. Relationship with monads. In an enriched situation, because $\underline{S} \multimap (-)$ is right adjoint to $!(-) \otimes \underline{S}$, any computation type \underline{S} induces a monad on value types:

$$\underline{S} \multimap !(-) \otimes \underline{S} \tag{1.1}$$

We call this a *linear-use state monad*. In Section 3.2 we show that every monad arises as a linear-use state monad. In brief, the argument is as follows. The right notion of monad for programming languages is ‘strong monad T on a category \mathbf{V} with finite products and Kleisli exponentials’; for every such monad T , the Kleisli category \mathbf{C} is \mathbf{V} -enriched and has copowers and is thus a model of the enriched call-by-value language. The object 1 in \mathbf{C} induces a linear-use state monad on \mathbf{V} (1.1) which is isomorphic to the original monad T on \mathbf{V} .

For the simplest case, consider the following monad on the category of sets for storing a single bit. If we let $\mathbf{V} = \mathbf{C} = \mathbf{Set}$ and let $\underline{S} = 2$ (there are two states), then the linear-use state monad is the usual state monad $TX = 2 \rightarrow (X \times 2)$.

In general, the linear-use state monad (1.1) arises from an adjunction that is parameterized in \underline{S} . Atkey [3] has investigated monads that arise from parameterized adjunctions: in fact Atkey’s parameterized monads are essentially the same as our enriched models. We return to this in Section 9.

1.4. Algebraic theories and state access operations. In order to explain the general connection between effects and state, we turn to the analysis of effects begun by Plotkin and Power [38]. Categorically, a finitary monad on the category of sets is essentially the same thing as an algebraic theory. Plotkin and Power proposed to use this connection to investigate computational effects from the perspective of universal algebra.

The analysis of Plotkin and Power centres around the correspondence between *algebraic operations* and *generic effects*. More precisely, for a monad T the following data are equivalent:

- (1) An *algebraic operation*: roughly, a functorial assignment of an n -ary function $X^n \rightarrow X$ to the carrier of each T -algebra $T(X) \rightarrow X$;
- (2) A *generic effect*: a function $1 \rightarrow T(n)$.

For instance, consider the store monad for a single bit of memory (using the isomorphic presentation $T = ((-) \times 2 \times (-) \times 2)$). Each T -algebra $\xi : T(X) \rightarrow X$ supports an algebraic operation $?_X : X \times X \rightarrow X$ in a functorial way: let $x ?_X y = \xi(x, 0, y, 1)$. If we understand elements of T -algebras as computations, then $x ?_X y$ is the computation that reads the bit in memory and then branches to either x or y depending on what was read.

The corresponding generic effect $\mathbf{deref} : 1 \rightarrow T(2)$ is given by $\mathbf{deref}() = (0, 0, 1, 1)$. It can be thought of as the command that reads the bit and returns the contents of memory.

We have already explained (§1.3) that all monads can be understood as linear-use state monads, $T = (\underline{S} \multimap (!(-) \otimes \underline{S}))$. The data for algebraic operations and generic effects can equivalently be given in terms of the following structure on the state object \underline{S} :

- (3) A *state access operation*: a morphism $\underline{S} \rightarrow !n \otimes \underline{S}$ in the computation category \mathbf{C} .

For instance, let $\mathbf{V} = \mathbf{C} = \mathbf{Set}$ and $\underline{S} = 2$. This gives a state monad for a single bit of memory, isomorphic to $((-) \times 2 \times (-) \times 2)$. The state access operation corresponding to reading the bit is simply the function $\text{read} : 2 \rightarrow 2 \times 2$ given by $\text{read}(i) = (i, i)$, which reads the store and returns the result along with the store.

In Section 8 we provide more examples and investigate a general framework for algebraic presentations of theories of effects using algebraic operations, generic effects and state access operations.

1.5. The enriched effect calculus. The work presented here grew out of work on the enriched effect calculus by Egger, Møgelberg and Simpson (EEC, [7, 8, 9, 10]). The enriched call-by-value calculus that we introduce in this paper is a fragment of EEC and our categorical semantics is based on their work. Every model of EEC contains a monad, and one of the observations of [9] (see also [7, Example 4.2]) was that this monad can always be represented as a linear state monad. A special case of the embedding theorem [8, Theorem 4] shows that given a strong monad T on a category \mathbf{V} with finite products and Kleisli exponentials we can embed \mathbf{V} in a model of EEC preserving the monad and all other structure. This gives the motto: *every monad embeds in a linear state monad*.

Since the enriched call-by-value calculus is a fragment of EEC (as opposed to full EEC), we allow ourselves a broader class of models. In contrast to the earlier work on EEC,

we do not include products among the computation types, since they are not needed in the state-passing translation, and so in our models the category \mathbf{C} does not need to have products. This allows us to build models from Kleisli categories, which typically do not have products, and this makes the relationship with monad models and closed Freyd categories much more straightforward. In particular, in our setting every monad *is* a linear state monad.

Nonetheless, in Section 10 we show that EEC is a conservative extension of the enriched call-by-value calculus. This shows that there is a fully-complete linear-use state translation into EEC. This result is further evidence that EEC is a promising calculus for reasoning about linear usage of effects. The related papers [9, 10] show how the linear-use continuation passing translation arises from a natural dual model construction on models of EEC, and use this to prove a full completeness theorem similar to that proven here for the linear-use state-passing translation. In fact, from the point of view of EEC the two translations are surprisingly similar: the linear-use state-passing translation is essentially dual to the linear-use continuation-passing translation. This observation goes back to the work on EEC and indeed duality plays a key role in [9] (although the relationship with state wasn't made explicit there). We draw it out explicitly in Section 7.

Acknowledgements. We thank Alex Simpson for help and encouragement. Also thanks to Lars Birkedal, Jeff Egger, Masahito Hasegawa, Shin-ya Katsumata and Paul Levy for helpful discussions. Diagrams are typeset using the `xymatrix` package and Paul Taylor's `diagrams` package.

2. ENRICHED CALL-BY-VALUE: A CALCULUS FOR ENRICHED CATEGORIES WITH COPOWERS

The target language for the linear-use state translation is a new calculus called the *enriched call-by-value calculus* (ECBV), that we now introduce. As we will explain, it is an internal language for enriched categories with copowers.

The enriched call-by-value calculus is a fragment of the enriched effect calculus (EEC), which was introduced by Egger et al. [8, 7] as a calculus for reasoning about linear usage in computational effects. The types of ECBV can be understood as a fragment of linear logic that is expressive enough to describe the linear state monad, $\underline{\mathbb{S}} \multimap !(-) \otimes \underline{\mathbb{S}}$. We will not dwell on the connection with linear logic here.

2.1. Type theory and equational theory. The enriched call-by-value calculus has two collections of types: value types and computation types. We use α, β, \dots to range over a set of *value type constants*, and $\underline{\alpha}, \underline{\beta}, \dots$ to range over a disjoint set of *computation type constants*. We then use upright letters A, B, \dots to range over value types, and underlined letters $\underline{A}, \underline{B}, \dots$ to range over computation types, which are specified by the grammar below:

$$\begin{aligned} A, B &::= \alpha \mid 1 \mid A \times B \mid \underline{A} \multimap \underline{B} \\ \underline{A}, \underline{B} &::= \underline{\alpha} \mid !A \otimes \underline{B} . \end{aligned}$$

Note that the construction $!A \otimes \underline{B}$ is indivisible: the strings $!\alpha$ and $\underline{\alpha} \otimes \underline{\beta}$ are not well-formed types. The stratification of types means that one cannot chain function types: the string $\underline{\alpha} \multimap (\underline{\beta} \multimap \underline{\gamma})$ is not well-formed.

Readers familiar with Levy's Call-by-Push-Value [26] or EEC [8] should note that there are no type constructors F and U for shifting between value and computation types and that computation types are not included in the value types. The only way to shift between value types and computation types is by using tensor and function types. As we will see, this is the essence of the state-passing translation.

The enriched call-by-value calculus has two basic typing judgements, written

$$\Gamma \mid - \vdash t : \mathbf{B} \quad \text{and} \quad \Gamma \mid z : \underline{\mathbf{A}} \vdash t : \underline{\mathbf{B}} \quad (2.1)$$

In the both judgements, Γ is an assignment of value types to variables. In the first judgement, \mathbf{B} is a value type, and in the second judgement, both $\underline{\mathbf{A}}$ and $\underline{\mathbf{B}}$ need to be computation types. The second judgement should be thought of as a judgement of linearity in the variable $z : \underline{\mathbf{A}}$. These judgements are defined inductively by the typing rules in Figure 1, which are a restriction of the rules of EEC [7] to this type structure. In the figure, Δ is an assignment of a computation type to a single variable, as in (2.1). The ideas behind the term language for the function space $\underline{\mathbf{A}} \multimap \underline{\mathbf{B}}$ and the tensor $!\underline{\mathbf{A}} \otimes \underline{\mathbf{B}}$ go back to the early work on linear lambda calculus. In particular, the introduction rule for $!\underline{\mathbf{A}} \otimes \underline{\mathbf{B}}$ uses pairing, and the elimination rule uses a pattern matching syntax.

2.2. Enriched call-by-value models. The categorical notion of model for ECBV involves basic concepts from enriched category theory [21]. In summary, a model of the language comprises two categories, \mathbf{V} and \mathbf{C} , interpreting the value and computation types respectively; the function type $\underline{\mathbf{A}} \multimap \underline{\mathbf{B}}$ provides an enrichment of \mathbf{C} in \mathbf{V} , and the universal property of the tensor type $!\underline{\mathbf{A}} \otimes \underline{\mathbf{B}}$ is the copower, sometimes called tensor.

We now make this precise. Let us recall some rudiments. Following [18, 14], we begin with actions of categories. Let \mathbf{V} be a category with finite products (by which we mean that it has a chosen terminal object and chosen binary products). Recall that an *action* of \mathbf{V} on a category \mathbf{C} is a functor $\cdot : \mathbf{V} \times \mathbf{C} \rightarrow \mathbf{C}$ together with two natural isomorphisms, unit ($1 \cdot \underline{\mathbf{A}} \cong \underline{\mathbf{A}}$) and associativity ($((\underline{\mathbf{A}} \times \underline{\mathbf{B}}) \cdot \underline{\mathbf{C}}) \cong (\underline{\mathbf{A}} \cdot (\underline{\mathbf{B}} \cdot \underline{\mathbf{C}}))$), that cohere with the isomorphisms arising from the product structure of \mathbf{V} in the following sense:

$$\begin{array}{ccc} (A \times 1) \cdot \underline{\mathbf{D}} & & (1 \times A) \cdot \underline{\mathbf{D}} \\ \cong \downarrow & \searrow \cong & \cong \downarrow \\ A \cdot (1 \cdot \underline{\mathbf{D}}) & \xrightarrow{\cong} & A \cdot \underline{\mathbf{D}} & & 1 \cdot (A \cdot \underline{\mathbf{D}}) & \xrightarrow{\cong} & A \cdot \underline{\mathbf{D}} \\ & & & & & & \\ ((A \times B) \times C) \cdot \underline{\mathbf{D}} & \xrightarrow{\cong} & (A \times B) \cdot (C \cdot \underline{\mathbf{D}}) \\ \cong \downarrow & & \downarrow \cong \\ (A \times (B \times C)) \cdot \underline{\mathbf{D}} & \xrightarrow{\cong} & A \cdot ((B \times C) \cdot \underline{\mathbf{D}}) & \xrightarrow{\cong} & A \cdot (B \cdot (C \cdot \underline{\mathbf{D}})) \end{array}$$

(We underline objects of \mathbf{C} to distinguish them from objects of \mathbf{V} .)

An *enrichment of a category \mathbf{C} in \mathbf{V} with copowers* is determined by an action of \mathbf{V} on \mathbf{C} such that each functor $(- \cdot \underline{\mathbf{B}}) : \mathbf{V} \rightarrow \mathbf{C}$ has a right adjoint, $\mathbf{C}(\underline{\mathbf{B}}, -) : \mathbf{C} \rightarrow \mathbf{V}$. Then $A \cdot \underline{\mathbf{B}}$ is called a copower, and $\mathbf{C}(\underline{\mathbf{B}}, \underline{\mathbf{C}})$ is called enrichment. We write $\text{Hom}_{\mathbf{C}}(\underline{\mathbf{B}}, \underline{\mathbf{C}})$ for the usual hom-set of \mathbf{C} to distinguish it from the enrichment.

Definition 2.1. An *enriched call-by-value model* (or simply *enriched model*) is given by a category \mathbf{V} with finite products and a category \mathbf{C} enriched in \mathbf{V} with copowers.

Types.

$$\begin{aligned} \mathbf{A}, \mathbf{B} &::= \alpha \mid 1 \mid \mathbf{A} \times \mathbf{B} \mid \mathbf{A} \multimap \mathbf{B} \\ \underline{\mathbf{A}}, \underline{\mathbf{B}} &::= \underline{\alpha} \mid !\mathbf{A} \otimes \underline{\mathbf{B}} . \end{aligned}$$

Term formation.

$$\begin{array}{c} \frac{}{\Gamma, x:A, \Gamma' \vdash x:A} \quad \frac{}{\Gamma \mid z:\underline{\mathbf{A}} \vdash z:\underline{\mathbf{A}}} \quad \frac{}{\Gamma \vdash \star:1} \\ \\ \frac{\Gamma \vdash t:A \quad \Gamma \vdash u:B}{\Gamma \vdash \langle t, u \rangle: \mathbf{A} \times \mathbf{B}} \quad \frac{\Gamma \vdash t: \mathbf{A}_1 \times \mathbf{A}_2}{\Gamma \vdash \pi_i(t): \mathbf{A}_i} \\ \\ \frac{\Gamma \mid z:\underline{\mathbf{A}} \vdash t:\underline{\mathbf{B}}}{\Gamma \vdash \underline{\lambda}z. t: \mathbf{A} \multimap \mathbf{B}} \quad \frac{\Gamma \vdash s: \mathbf{A} \multimap \mathbf{B} \quad \Gamma \mid \Delta \vdash t:\underline{\mathbf{A}}}{\Gamma \mid \Delta \vdash s[t]: \underline{\mathbf{B}}} \\ \\ \frac{\Gamma \vdash t:A \quad \Gamma \mid \Delta \vdash u:\underline{\mathbf{B}}}{\Gamma \mid \Delta \vdash !t \otimes u: !\mathbf{A} \otimes \underline{\mathbf{B}}} \quad \frac{\Gamma \mid \Delta \vdash t: !\mathbf{A} \otimes \underline{\mathbf{B}} \quad \Gamma, x:A \mid z:\underline{\mathbf{B}} \vdash u:\underline{\mathbf{C}}}{\Gamma \mid \Delta \vdash t \text{ to } (!x \otimes z). u: \underline{\mathbf{C}}} \end{array}$$

Equality. (We elide α -equivalence, reflexivity, symmetry, transitivity and congruence laws.)

$$\begin{array}{c} \frac{\Gamma \vdash t:1}{\Gamma \vdash t \equiv \star:1} \quad \frac{\Gamma \vdash t_1: \mathbf{A}_1 \quad \Gamma \vdash t_2: \mathbf{A}_2}{\Gamma \vdash \pi_i(\langle t_1, t_2 \rangle) \equiv t_i: \mathbf{A}_i} \\ \\ \frac{\Gamma \vdash t: \mathbf{A}_1 \times \mathbf{A}_2}{\Gamma \vdash \langle \pi_1(t), \pi_2(t) \rangle \equiv t: \mathbf{A}_1 \times \mathbf{A}_2} \\ \\ \frac{\Gamma \mid z:\underline{\mathbf{A}} \vdash t:\underline{\mathbf{B}} \quad \Gamma \mid \Delta \vdash u:\underline{\mathbf{A}}}{\Gamma \mid \Delta \vdash (\underline{\lambda}z. t)[u] \equiv t[u/z]: \underline{\mathbf{B}}} \quad \frac{\Gamma \vdash t: \mathbf{A} \multimap \mathbf{B}}{\Gamma \vdash t \equiv \underline{\lambda}z. (t[z]): \mathbf{A} \multimap \mathbf{B}} \\ \\ \frac{\Gamma \vdash t:A \quad \Gamma \mid \Delta \vdash u:\underline{\mathbf{B}} \quad \Gamma, x:A \mid z:\underline{\mathbf{B}} \vdash v:\underline{\mathbf{C}}}{\Gamma \mid \Delta \vdash (!t \otimes u) \text{ to } (!x \otimes z). v \equiv v[t/x, u/z]: \underline{\mathbf{C}}} \\ \\ \frac{\Gamma \mid \Delta \vdash t: !\mathbf{A} \otimes \underline{\mathbf{B}} \quad \Gamma \mid y: !\mathbf{A} \otimes \underline{\mathbf{B}} \vdash u:\underline{\mathbf{C}}}{\Gamma \mid \Delta \vdash t \text{ to } (!x \otimes z). u[!x \otimes z/y] \equiv u[t/y]: \underline{\mathbf{C}}} \end{array}$$

Figure 1: The enriched call-by-value calculus

In Section 2.3 we will illustrate the definition with some examples of enriched models. First, let us clarify the semantics for ECBV in an enriched model. The interpretation is similar to the semantics of EEC proposed by Egger et al. [8].

- A value type A is interpreted as an object $\llbracket A \rrbracket$ of \mathbf{V} , and a computation type \underline{A} is interpreted as an object $\llbracket \underline{A} \rrbracket$ of \mathbf{C} , as follows. The interpretation is defined by induction on the structure of types. First, for each value type constant α , an object $\llbracket \alpha \rrbracket$ of \mathbf{V} is given, and for each computation type constant $\underline{\alpha}$ an object $\llbracket \underline{\alpha} \rrbracket$ of \mathbf{C} is given. The product types are interpreted as products in \mathbf{V} . The remaining type constructions are interpreted using the enriched structure: we let $\llbracket !A \otimes B \rrbracket \stackrel{\text{def}}{=} (\llbracket A \rrbracket \cdot \llbracket B \rrbracket)$, and $\llbracket A \multimap B \rrbracket \stackrel{\text{def}}{=} \mathbf{C}(\llbracket A \rrbracket, \llbracket B \rrbracket)$.
- A value context $\Gamma = (x_1 : A_1, \dots, x_n : A_n)$ is interpreted as a product $\llbracket A_1 \rrbracket \times \dots \times \llbracket A_n \rrbracket$ in \mathbf{V} . A computation context $\Delta = (z : \underline{A})$ is interpreted as the object $\llbracket \underline{A} \rrbracket$ in \mathbf{C} .
- A judgement $\Gamma \mid - \vdash t : A$ is interpreted as a morphism $\llbracket \Gamma \rrbracket \rightarrow \llbracket A \rrbracket$ in \mathbf{V} , and a judgement $\Gamma \mid \Delta \vdash t : \underline{A}$ is interpreted as a morphism $\llbracket \Gamma \rrbracket \cdot \llbracket \Delta \rrbracket \rightarrow \llbracket \underline{A} \rrbracket$ in \mathbf{C} . This definition is made by induction on the structure of typing derivations, making use of the universal properties of the interpretations of the types. For illustration, we consider the following two rules:

$$\frac{\Gamma \mid z : \underline{A} \vdash t : \underline{B}}{\Gamma \mid - \vdash \lambda z. t : \underline{A} \multimap \underline{B}} \quad \frac{\Gamma \mid \Delta \vdash t : !A \otimes \underline{B} \quad \Gamma, x : A \mid z : \underline{B} \vdash u : \underline{C}}{\Gamma \mid \Delta \vdash t \text{ to } (!x \otimes z). u : \underline{C}}$$

In dealing with the linear lambda abstraction rule, the induction principle gives us an interpretation $\llbracket t \rrbracket : \llbracket \Gamma \rrbracket \cdot \llbracket \underline{A} \rrbracket \rightarrow \llbracket \underline{B} \rrbracket$ in \mathbf{C} which we use to form $\llbracket \lambda z. t \rrbracket : \llbracket \Gamma \rrbracket \rightarrow \mathbf{C}(\llbracket \underline{A} \rrbracket, \llbracket \underline{B} \rrbracket)$ in \mathbf{V} , using the natural bijection that defines the relationship between the copower and the enrichment:

$$\text{Hom}_{\mathbf{C}}(A \cdot \underline{B}, \underline{C}) \cong \text{Hom}_{\mathbf{V}}(A, \mathbf{C}(\underline{B}, \underline{C})).$$

For the pattern matching rule, we assume morphisms

$$\llbracket t \rrbracket : \llbracket \Gamma \rrbracket \cdot \llbracket \Delta \rrbracket \rightarrow \llbracket A \rrbracket \cdot \llbracket B \rrbracket \quad \llbracket u \rrbracket : (\llbracket \Gamma \rrbracket \times \llbracket A \rrbracket) \cdot \llbracket B \rrbracket \rightarrow \llbracket C \rrbracket$$

in \mathbf{C} and use them to define $\llbracket t \text{ to } (!x \otimes z). u \rrbracket : \llbracket \Gamma \rrbracket \cdot \llbracket \Delta \rrbracket \rightarrow \llbracket C \rrbracket$ as the following composite:

$$\begin{aligned} \llbracket \Gamma \rrbracket \cdot \llbracket \Delta \rrbracket &\xrightarrow{\text{diag.}} (\llbracket \Gamma \rrbracket \times \llbracket \Gamma \rrbracket) \cdot \llbracket \Delta \rrbracket \xrightarrow{\cong} \llbracket \Gamma \rrbracket \cdot (\llbracket \Gamma \rrbracket \cdot \llbracket \Delta \rrbracket) \xrightarrow{\llbracket \Gamma \rrbracket \cdot \llbracket t \rrbracket} \llbracket \Gamma \rrbracket \cdot (\llbracket A \rrbracket \cdot \llbracket B \rrbracket) \\ &\xrightarrow{\cong} (\llbracket \Gamma \rrbracket \times \llbracket A \rrbracket) \cdot \llbracket B \rrbracket \xrightarrow{\llbracket u \rrbracket} \llbracket C \rrbracket \end{aligned}$$

Proposition 2.2. *The interpretation of the enriched call-by-value calculus in an enriched model (\mathbf{V}, \mathbf{C}) is sound:*

- (1) *If $\Gamma \mid - \vdash t \equiv u : A$ then $\llbracket t \rrbracket = \llbracket u \rrbracket : \llbracket \Gamma \rrbracket \rightarrow \llbracket A \rrbracket$ in \mathbf{V} .*
- (2) *If $\Gamma \mid \Delta \vdash t \equiv u : \underline{A}$ then $\llbracket t \rrbracket = \llbracket u \rrbracket : \llbracket \Gamma \rrbracket \cdot \llbracket \Delta \rrbracket \rightarrow \llbracket \underline{A} \rrbracket$ in \mathbf{C} .*

Proof notes. This is proved by induction on the structure of the derivation of (\equiv) . The following substitution lemma is helpful:

If $\Gamma \mid \Delta \vdash t : \underline{A}$ and $\Gamma \mid z : \underline{A} \vdash u : \underline{B}$ then $u[t/z]$ is the following composite morphism in \mathbf{C} :

$$\llbracket \Gamma \rrbracket \cdot \llbracket \Delta \rrbracket \xrightarrow{\text{diag.}} (\llbracket \Gamma \rrbracket \times \llbracket \Gamma \rrbracket) \cdot \llbracket \Delta \rrbracket \xrightarrow{\cong} \llbracket \Gamma \rrbracket \cdot (\llbracket \Gamma \rrbracket \cdot \llbracket \Delta \rrbracket) \xrightarrow{\llbracket \Gamma \rrbracket \cdot \llbracket t \rrbracket} \llbracket \Gamma \rrbracket \cdot \llbracket \underline{A} \rrbracket \xrightarrow{\llbracket u \rrbracket} \llbracket \underline{B} \rrbracket \quad \square$$

2.3. Examples of enriched models. We now list some examples of enriched models (Definition 2.1).

- (1) If $\mathbf{V} = \mathbf{Set}$ then a \mathbf{V} -enriched category is just a locally small category. The copower $A \cdot \underline{B}$ is the A -fold iterated coproduct of \underline{B} , if it exists. The following three examples are instances of this example.
- (2) Let $\mathbf{V} = \mathbf{Set}$ and let \mathbf{C} be the category of monoids and homomorphisms. The copower $A \cdot \underline{B}$, where A is a set and \underline{B} is a monoid, can be described as a quotient of the free monoid on the product of sets, $(A \times |\underline{B}|)^* / \sim$. Here $(A \times |\underline{B}|)^*$ is the set of strings built of pairs in $(A \times |\underline{B}|)$, which is a monoid under concatenation with the empty string ϵ as unit. The equivalence relation (\sim) is generated by $(a, b).(a, b') \sim (a, b.b')$ and $\epsilon \sim (a, \epsilon)$. There is a similar description of the copower for any algebraic theory.
- (3) We can modify Example (2) to make \mathbf{C} the category of *free monoids* and monoid homomorphisms. That is, the objects are monoids of the form B^* . In this situation, the copower satisfies $A \cdot B^* = (A \times B)^*$. In this example \mathbf{C} is the Kleisli category for the free monoid monad. We will revisit Kleisli categories in Section 3.2.
- (4) Let $\mathbf{V} = \mathbf{C} = \mathbf{Set}$, with \mathbf{C} considered with the ordinary enrichment. The copower $A \cdot B$ is the cartesian product of sets. This situation generalizes to the situation where $\mathbf{V} = \mathbf{C}$ is an arbitrary cartesian closed category.
- (5) Let \mathbf{V} be the category of ω -cpo's and continuous functions, and let \mathbf{C} be the category of pointed ω -cpo's and strict functions. The enrichment $\mathbf{C}(\underline{A}, \underline{B})$ is the cpo of strict functions under the pointwise order, and the copower $A \cdot \underline{B}$ is the smash product $A_{\perp} \otimes \underline{B}$.
- (6) In the next section we will investigate a model built from the syntax of the enriched call-by-value calculus.

2.4. The syntactic enriched model. The types and terms of the enriched call-by-value calculus form an enriched model which we call the syntactic enriched model.

Let \mathbf{V} be the category whose objects are value types and where a morphism $A \rightarrow B$ is a term in context $x: A \mid - \vdash t: B$ modulo the equational theory (Figure 1) and modulo renaming the free variable x . The identity morphism is the term $x: A \mid - \vdash x: A$, and composition of morphisms

$$A \xrightarrow{x: A \mid - \vdash t: B} B \xrightarrow{y: B \mid - \vdash u: C} C$$

is given by substitution: $u \circ t \stackrel{\text{def}}{=} (x: A \mid - \vdash u[t/y]): C$. Since morphisms are actually equivalence classes, the well-definedness of substitution depends on the following substitution lemma

$$\begin{aligned} & \text{If } x: A \mid - \vdash t \equiv t': B \text{ and } y: B \mid - \vdash u \equiv u': C \\ & \text{then } x: A \mid - \vdash u[t/y] \equiv u'[t'/y]: C \end{aligned}$$

which is proved by induction on the derivation of $u \equiv u'$.

The laws of associativity and identity for composition are immediate. For instance, associativity amounts to the following syntactic identity:

$$\begin{aligned} & \text{If } x: A \mid - \vdash t: B, \quad y: B \mid - \vdash u: C \text{ and } z: C \mid - \vdash v: D \\ & \text{then } x: A \mid - \vdash (v[u/z])[t/y] \equiv v[u[t/y]/z]: D. \end{aligned}$$

The category \mathbf{V} has products, given by the product types. The equations at product types are exactly what is needed to guarantee the universal properties.

Let \mathbf{C} be the category whose objects are computation types and where a morphism $\underline{A} \rightarrow \underline{B}$ is a term in context $- \mid z: \underline{A} \vdash t: \underline{B}$ modulo the equational theory and modulo renaming the free variable z . Identities and composition are defined in a similar way to \mathbf{V} . The identity morphisms $\underline{A} \rightarrow \underline{A}$ are $- \mid z: \underline{A} \vdash z: \underline{A}$ and composition is by substitution.

The action of \mathbf{V} on \mathbf{C} is given on objects by the tensor type: let $\underline{A} \cdot \underline{B} \stackrel{\text{def}}{=} !\underline{A} \otimes \underline{B}$. Given morphisms

$$\underline{A} \xrightarrow{x: \underline{A} \vdash t: \underline{A}'} \underline{A}' \text{ in } \mathbf{V} \quad \text{and} \quad \underline{B} \xrightarrow{- \mid z: \underline{B} \vdash u: \underline{B}'} \underline{B}' \text{ in } \mathbf{C}$$

we define a morphism $t \cdot u: (\underline{A} \cdot \underline{B}) \rightarrow (\underline{A}' \cdot \underline{B}')$ in \mathbf{C} by

$$t \cdot u \stackrel{\text{def}}{=} (- \mid z': !\underline{A} \otimes \underline{B} \vdash z' \text{ to } (!x \otimes z). !t \otimes u: !\underline{A}' \otimes \underline{B}').$$

Functoriality follows from the equational theory of ECBV. The unit and associativity isomorphisms are straightforward to define. For example, associativity $((\underline{A} \times \underline{B}) \cdot \underline{C} \cong \underline{A} \cdot (\underline{B} \cdot \underline{C}))$ is given by exhibiting mutual inverses at all types:

$$\begin{aligned} & - \mid z: !(\underline{A} \times \underline{B}) \otimes \underline{C} \vdash z \text{ to } (!x \otimes z'). !(\pi_1 x) \otimes (!(\pi_2 x) \otimes z'): !\underline{A} \otimes (!\underline{B} \otimes \underline{C}) \\ & - \mid z: !\underline{A} \otimes (!\underline{B} \otimes \underline{C}) \vdash z \text{ to } (!x \otimes z'). z' \text{ to } (!y \otimes z''). !(x, y) \otimes z'': !(\underline{A} \times \underline{B}) \otimes \underline{C} \end{aligned}$$

It follows from the equational theory of ECBV that these are isomorphisms and are natural and coherent.

Finally, we discuss the enrichment of \mathbf{C} in \mathbf{V} . Given types \underline{A} , \underline{B} and \underline{C} we describe a natural bijection

$$\text{Hom}_{\mathbf{C}}(\underline{A} \cdot \underline{B}, \underline{C}) \cong \text{Hom}_{\mathbf{V}}(\underline{A}, \underline{B} \multimap \underline{C})$$

From left to right the bijection takes a computation term $- \mid z: !\underline{A} \otimes \underline{B} \vdash t: \underline{C}$ to a value term $x: \underline{A} \mid - \vdash \lambda b. t^{[(!x \otimes b)/z]}: \underline{B} \multimap \underline{C}$. From right to left the bijection takes a value term $x: \underline{A} \mid - \vdash u: \underline{B} \multimap \underline{C}$ to a computation term $- \mid z: (!\underline{A} \otimes \underline{B}) \vdash z \text{ to } (!x \otimes y). u[y]: \underline{C}$.

2.5. Universal property of the syntactic enriched model. The syntactic model described in Section 2.4 enjoys a universal property: it is an initial object in a category of enriched models with structure preserving functors as morphisms. Given any other enriched model, the unique morphism from the syntactic model is given by interpretation of syntax in the model.

This semantic characterization of interpretation is standard in categorical semantics, and it is useful for deriving syntactic results from semantics, as we shall see in Section 5. However, we shall also see (Section 5.3) that we need to talk about morphisms of models preserving structure only up to isomorphism, and the syntactic model is not initial with respect to this collection of morphisms. Rather, interpretation defines a morphism which is unique only up to unique isomorphism. In order to formulate this kind of initiality, we need to be able to assess when two morphisms between enriched models are isomorphic. Thus the enriched models form a groupoid-enriched category, i.e., a 2-category in which each 2-cell is invertible. The idea of using groupoid-enriched categories of models has been around for a long time (e.g. [6, §8]) and has been used in previous work on the enriched effect calculus [8, 9].

A precise definition of the 2-category of enriched models \mathfrak{Entr} can be found in Appendix A. Here we just state the initiality property of the syntactic model. First recall the following definition from 2-category theory (see e.g. [22, §6]).

Definition 2.3. Let \mathfrak{K} be a 2-category. An object 0 of \mathfrak{K} is *bi-initial* if for any object A the hom-category $\mathfrak{K}(0, A)$ is equivalent to the terminal category (i.e., the category with one object and one morphism).

An equivalent way of stating bi-initiality is to say that for any other object A there exists a 1-cell $0 \rightarrow A$ which is unique up to unique 2-isomorphism.

The syntactic model of Section 2.4 is bi-initial in the category \mathbf{Ent} , but in this paper we are much more interested in a category of enriched models (\mathbf{V}, \mathbf{C}) with a specified state object \underline{S} in \mathbf{C} (because of the relation to the state-passing translation), and so we formulate bi-initiality with respect to a category \mathbf{Ecbv} of these. Like all other structure, 1-cells of \mathbf{Ecbv} are only required to preserve the state objects up to isomorphism. (See Appendix A.2 for a full definition). We write $(\mathcal{V}_{\text{ECBV}}, \mathcal{C}_{\text{ECBV}}, \underline{S})$ for the enriched model obtained as in Section 2.4 from the syntax of the enriched call-by-value calculus extended with a special computation type constant \underline{S} .

Theorem 2.4. *The model $(\mathcal{V}_{\text{ECBV}}, \mathcal{C}_{\text{ECBV}}, \underline{S})$ is bi-initial in \mathbf{Ecbv} . The unique morphisms with domain $(\mathcal{V}_{\text{ECBV}}, \mathcal{C}_{\text{ECBV}}, \underline{S})$ are given by interpretation of syntax into models.*

3. FINE-GRAIN CALL-BY-VALUE, A CALCULUS FOR KLEISLI MODELS

The source language for our state-passing translation is a call-by-value language equipped with an equational theory to be thought of as generated by some operational semantics, as in [34]. We use a ‘fine grain’ call-by-value language, following Levy et al. [27, 26]. We use α to range over type constants. The types are given by the grammar

$$\sigma, \tau ::= \alpha \mid 1 \mid \sigma \times \tau \mid \sigma \rightarrow \tau.$$

The function space \rightarrow is a call-by-value one, which takes a value and produces a computation.

The fine-grain call-by-value calculus (FGCBV) has two typing judgements, one for values and one for producers. These are written $\Gamma \vdash^v V : \sigma$ and $\Gamma \vdash^p M : \sigma$. The latter should be thought of as typing computations which produce values in the type judged but may also perform side-effects along the way. In both judgements the variables of the contexts are to be considered as placeholders for values. Typing rules along with equality rules are given in Figure 2.

The call-by-value language is called ‘fine grain’ because the order of evaluation is explicit. Notice that the string $(f(x), g(y))$ is not well-formed syntax: one must specify the order of evaluation, for instance, like this:

$$f(x) \text{ to } x'. g(y) \text{ to } y'. (x', y').$$

Translations from a ‘coarser grain’, more natural programming language are given by Levy et al. ([27, §3], [26, §A.3.3]).

3.1. Interpretation in Kleisli models. The most natural way to interpret fine-grain call-by-value is to have two categories \mathbf{V} and \mathbf{C} to interpret the judgements \vdash^v and \vdash^p , but to insist that the two categories have exactly the same objects, since in this language there is only one class of types.

<p><i>Types.</i></p> $\sigma, \tau ::= \alpha \mid 1 \mid \sigma \times \tau \mid \sigma \multimap \tau.$ <hr/> <p><i>Term formation.</i></p> $\frac{}{\Gamma, x: \sigma, \Gamma' \vdash^v x: \sigma} \quad \frac{}{\Gamma \vdash^v \star: 1} \quad \frac{\Gamma \vdash^v V: \sigma_1 \times \sigma_2}{\Gamma \vdash^v \pi_i(V): \sigma_i} \quad \frac{\Gamma \vdash^v V_1: \sigma_1 \quad \Gamma \vdash^v V_2: \sigma_2}{\Gamma \vdash^v \langle V_1, V_2 \rangle: \sigma_1 \times \sigma_2}$ $\frac{\Gamma \vdash^v V: \sigma}{\Gamma \vdash^P \text{return } V: \sigma} \quad \frac{\Gamma \vdash^P M: \sigma \quad \Gamma, x: \sigma \vdash^P N: \tau}{\Gamma \vdash^P M \text{ to } x. N: \tau}$ $\frac{\Gamma, x: \sigma \vdash^P N: \tau}{\Gamma \vdash^v \lambda x. N: \sigma \multimap \tau} \quad \frac{\Gamma \vdash^v V: \sigma \multimap \tau \quad \Gamma \vdash^v W: \sigma}{\Gamma \vdash^P V W: \tau}$ <hr/> <p><i>Equality.</i> (We elide α-equivalence, reflexivity, symmetry, transitivity and congruence laws.)</p> $\frac{\Gamma \vdash^v M: 1}{\Gamma \vdash^v M \equiv \star: 1} \quad \frac{\Gamma \vdash^v V_1: \sigma_2 \quad \Gamma \vdash^v V_2: \sigma_2}{\Gamma \vdash^v \pi_i(\langle V_1, V_2 \rangle) \equiv V_i: \sigma_i} \quad \frac{\Gamma \vdash^v V: \sigma_1 \times \sigma_2}{\Gamma \vdash^v \langle \pi_1(V), \pi_2(V) \rangle \equiv V: \sigma_1 \times \sigma_2}$ $\frac{\Gamma, x: \sigma \vdash^P M: \tau \quad \Gamma \vdash^v V: \sigma}{\Gamma \vdash^v (\lambda x. M) V \equiv M[V/x]: \tau} \quad \frac{\Gamma \vdash^v V: \sigma \multimap \tau}{\Gamma \vdash^v \lambda x. (V x) \equiv V: \sigma \multimap \tau}$ $\frac{\Gamma \vdash^P M: \sigma}{\Gamma \vdash^P M \text{ to } x. \text{return } x \equiv M: \sigma} \quad \frac{\Gamma \vdash^v V: \sigma \quad \Gamma, x: \sigma \vdash^P N: \tau}{\Gamma \vdash^P \text{return } V \text{ to } x. N \equiv N[V/x]: \tau}$ $\frac{\Gamma \vdash^P M: \sigma \quad \Gamma, x: \sigma \vdash^P N: \tau \quad \Gamma, y: \tau \vdash^P P: v}{\Gamma \vdash^P (M \text{ to } x. N) \text{ to } y. P \equiv M \text{ to } x. (N \text{ to } y. P): v}$	
---------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------	--

Figure 2: Fine-grain call-by-value.

Definition 3.1. An *enriched Kleisli model* is an enriched call-by-value model (\mathbf{V}, \mathbf{C}) (Def. 2.1) together with an identity-on-objects functor $J: \mathbf{V} \rightarrow \mathbf{C}$ that strictly preserves copowers, which means that $J(A \times B) = A \cdot J(B)$ (naturally in A and B) and that the canonical isomorphisms induced by the product structure are the coherent unit and associativity isomorphisms of the copowers:

$$1 \cdot JA = J(1 \times A) \cong JA \quad (A \times B) \cdot JC = J((A \times B) \times C) \cong J(A \times (B \times C)) = A \cdot (B \cdot JC).$$

We will sometimes say ‘Kleisli model’ for ‘enriched Kleisli model’. We use the name ‘Kleisli’ because this definition captures the situation where \mathbf{C} is the Kleisli category for a strong monad on \mathbf{V} . The correspondence is explained in Section 3.2.

Kleisli models have earlier been called ‘closed Freyd categories’ by Levy et al. [27]. Their original definition of closed Freyd category is based on premonoidal categories; the relationship with actions of categories and Kleisli models is observed by Levy [26, B.10].

A semantics for FGCBV is given in a Kleisli model in a standard way.

- Each base type is given an interpretation $\llbracket \alpha \rrbracket$ as an object of \mathbf{V} . This interpretation is extended to all types: $\llbracket 1 \rrbracket$ is the terminal object of \mathbf{V} ; $\llbracket \sigma \times \tau \rrbracket$ is the product of $\llbracket \sigma \rrbracket$ and $\llbracket \tau \rrbracket$; and $\llbracket \sigma \multimap \tau \rrbracket$ is defined using the enriched structure of \mathbf{C} : $\llbracket \sigma \multimap \tau \rrbracket \stackrel{\text{def}}{=} \mathbf{C}(\llbracket \sigma \rrbracket, \llbracket \tau \rrbracket)$.
- A context $\Gamma = (x_1 : \sigma_1, \dots, x_n : \sigma_n)$ is interpreted in \mathbf{V} as a product $\llbracket \sigma_1 \rrbracket \times \dots \times \llbracket \sigma_n \rrbracket$.
- A value type judgement $\Gamma \vdash^v V : \sigma$ is interpreted as a morphism $\llbracket \Gamma \rrbracket \rightarrow \llbracket \sigma \rrbracket$ in \mathbf{V} and a producer type judgement $\Gamma \vdash^p M : \sigma$ is interpreted as a morphism $\llbracket \Gamma \rrbracket \rightarrow \llbracket \sigma \rrbracket$ in \mathbf{C} . This is defined by induction on the structure of derivations, using the universal properties of Kleisli models. For illustration we consider the following rule.

$$\frac{\Gamma \vdash^p M : \sigma \quad \Gamma, x : \sigma \vdash^p N : \tau}{\Gamma \vdash^p M \text{ to } x. N : \tau}$$

The induction hypothesis gives us two morphisms in \mathbf{C}

$$\llbracket \Gamma \rrbracket \xrightarrow{\llbracket M \rrbracket} \llbracket \sigma \rrbracket \quad \llbracket \Gamma \rrbracket \times \llbracket \sigma \rrbracket \xrightarrow{\llbracket N \rrbracket} \llbracket \tau \rrbracket$$

and we use these to define a morphism in \mathbf{C} that interprets $(M \text{ to } x. N)$:

$$\llbracket \Gamma \rrbracket \xrightarrow{J(\text{diag})} \llbracket \Gamma \rrbracket \times \llbracket \Gamma \rrbracket \xrightarrow{\cong} \llbracket \Gamma \rrbracket \cdot \llbracket \Gamma \rrbracket \xrightarrow{\llbracket \Gamma \rrbracket \cdot \llbracket M \rrbracket} \llbracket \Gamma \rrbracket \cdot \llbracket \sigma \rrbracket \xrightarrow{\cong} \llbracket \Gamma \rrbracket \times \llbracket \sigma \rrbracket \xrightarrow{\llbracket N \rrbracket} \llbracket \tau \rrbracket.$$

As another example $\llbracket \text{return } V \rrbracket = J(\llbracket V \rrbracket)$.

This defines a sound and complete notion of model for FGCBV.

Proposition 3.2 ([27], Prop. 5.1). *The interpretation of the fine-grain call-by-value calculus in a Kleisli model is sound:*

- (1) If $\Gamma \vdash^v V \equiv W : \sigma$ then $\llbracket V \rrbracket = \llbracket W \rrbracket : \llbracket \Gamma \rrbracket \rightarrow \llbracket \sigma \rrbracket$ in \mathbf{V} .
- (2) If $\Gamma \vdash^p M \equiv N : \sigma$ then $\llbracket M \rrbracket = \llbracket N \rrbracket : \llbracket \Gamma \rrbracket \rightarrow \llbracket \sigma \rrbracket$ in \mathbf{C} .

3.2. Relationship with monads. We now explain the connection between enriched Kleisli models and Kleisli categories for a monad. For more detail, see the paper by Levy et al. [27].

From the syntactic point of view, the fine-grain call-by-value language can be thought of as a variant of Moggi’s λ_C [31]: the type construction $(1 \multimap (-))$ is a monad.

From the semantic point of view, recall that a λ_C -model [31] is a category with finite products and a strong monad with Kleisli exponentials. We now explain these conditions.

Let \mathbf{V} be a category with finite products, and let (T, η, μ) be a monad on \mathbf{V} . Let \mathbf{C} be the Kleisli category for T : the objects of \mathbf{C} are the objects of \mathbf{V} and a morphism $A \rightarrow B$ in \mathbf{C} is a morphism $A \rightarrow T(B)$ in \mathbf{V} . There is an identity-on-objects functor $J : \mathbf{V} \rightarrow \mathbf{C}$ which takes a morphism $f : A \rightarrow B$ in \mathbf{V} to the morphism $(\eta_B \cdot f) : A \rightarrow B$ in \mathbf{C} .

A strength for a monad T is usually expressed as a family of morphisms $A \times T(B) \rightarrow T(A \times B)$ that respect the structure of the monad. In fact, a monad is strong if and only if there is an action of \mathbf{V} on \mathbf{C} and the identity-on-objects functor $J : \mathbf{V} \rightarrow \mathbf{C}$ preserves it. The strength is needed to take a morphism $f : B \rightarrow B'$ in \mathbf{C} to a morphism $(A \cdot f) : A \cdot B \rightarrow A \cdot B'$ in \mathbf{C} .

The requirement of Kleisli exponentials is normally expressed as the requirement that for all A and B , the hom-functor $\text{Hom}_{\mathbf{V}}((-) \times A, TB) : \mathbf{V}^{\text{op}} \rightarrow \mathbf{Set}$ is representable. To

endow \mathbf{V} with Kleisli exponentials is to give a right adjoint for the action, i.e. an enrichment of \mathbf{C} in \mathbf{V} .

Conversely, every enriched Kleisli model $(\mathbf{V}, \mathbf{C}, J)$ induces a strong monad on \mathbf{V} with Kleisli exponentials. The monad is defined using the closed structure: $T(A) \stackrel{\text{def}}{=} \mathbf{C}(1, A)$. The Kleisli category for this monad is isomorphic to \mathbf{C} . On the other hand, if we begin with a monad, build the Kleisli category and then take the monad $\mathbf{C}(1, A)$, we recover a monad that is isomorphic to the one that we started with. In this sense, enriched Kleisli models and λ_C -models are equivalent. Note that they are not exactly the same, for although the hom-set $\text{Hom}_{\mathbf{V}}(A, \mathbf{C}(1, B))$ is in bijection with $\text{Hom}_{\mathbf{C}}(A, B)$, the sets are typically not identical.

3.3. The syntactic Kleisli model. The types and terms of the fine-grain call-by-value calculus form a syntactic model. We employ the same kinds of technique as for enriched call-by-value in Section 2.4.

- The objects of both \mathbf{V} and \mathbf{C} are the types of FGCBV.
- A morphism $\sigma \rightarrow \tau$ in \mathbf{V} is a value judgement $x: \sigma \vdash^v V: \tau$ modulo the equational theory \equiv (Figure 2) and modulo renaming the free variable x . Identities are variables and composition is by substitution.
- A morphism $\sigma \rightarrow \tau$ in \mathbf{C} is a computation judgement $x: \sigma \vdash^p M: \tau$ modulo the equational theory \equiv and renaming the free variable x . The identity $\sigma \rightarrow \sigma$ is $x: \sigma \vdash^p \mathbf{return} x: \sigma$. Composition is *not* by substitution, since one cannot substitute a producer term for a variable. Rather, the composite of

$$\sigma \xrightarrow{x: \sigma \vdash^p M: \tau} \tau \xrightarrow{y: \tau \vdash^p N: \upsilon} \upsilon$$

in \mathbf{C} is $x: \sigma \vdash^p M \mathbf{to} y. N: \upsilon$.

- The product structure in \mathbf{V} is given by the product types, projections and pairing.
- The action of \mathbf{V} on \mathbf{C} is given on objects by the binary product types: let $\sigma \cdot \tau \stackrel{\text{def}}{=} \sigma \times \tau$. On morphisms, given $A \xrightarrow{x: \sigma \vdash^v V: \sigma'} \sigma'$ in \mathbf{V} and $\tau \xrightarrow{y: \tau \vdash^p M: \tau'} \tau'$ in \mathbf{C} , we define

$$(\sigma \cdot \tau \xrightarrow{V \cdot M} \sigma' \cdot \tau') \stackrel{\text{def}}{=} z: \sigma \times \tau \vdash^p M[\pi_2(z)/y] \mathbf{to} y'. \mathbf{return} \langle V[\pi_1(z)/x], y' \rangle: \sigma' \times \tau'$$

- The enrichment is given by $\mathbf{C}(\sigma, \tau) \stackrel{\text{def}}{=} (\sigma \rightarrow \tau)$.
- The identity-on-objects functor $J: \mathbf{V} \rightarrow \mathbf{C}$ takes a morphism $\sigma \xrightarrow{x: \sigma \vdash^v V: \tau} \tau$ in \mathbf{V} to the morphism $\sigma \xrightarrow{x: \sigma \vdash^p \mathbf{return} V: \tau} \tau$ in \mathbf{C} .

We write $(\mathcal{V}_{\text{FGCBV}}, \mathcal{C}_{\text{FGCBV}}, J)$ for the syntactic Kleisli model.

3.4. Universal property of the syntactic Kleisli model. Appendix A.3 defines the 2-category $\mathfrak{Kleisli}$ of Kleisli models. As was the case for \mathfrak{Ecbv} , 1-cells are only required to preserve structure up to isomorphism.

Theorem 3.3. *The syntactic Kleisli model $(\mathcal{V}_{\text{FGCBV}}, \mathcal{C}_{\text{FGCBV}}, J)$ is bi-initial in $\mathfrak{Kleisli}$. The unique morphisms with domain $(\mathcal{V}_{\text{FGCBV}}, \mathcal{C}_{\text{FGCBV}}, J)$ are given by interpretation of syntax into models.*

4. THE LINEAR-USE STATE-PASSING TRANSLATION

This section defines the linear-use state-passing translation from the fine-grain call-by-value calculus to the enriched call-by-value calculus, and states the main syntactic results of this paper: fullness on types and full completeness. Together these assert that the linear-use state-passing translation is an equivalence of languages.

We now fix a computation type \underline{S} of ECBV. For now, it can be an arbitrary computation type; later we will make it a distinguished basic type to achieve a full completeness result. We will describe a translation from FGCBV to ECBV. When \underline{S} is thought of as a type of states, then this translation reads as a state-passing translation.

We translate FGCBV types σ to ECBV value types $\sigma^{\underline{S}}$:

$$\alpha^{\underline{S}} \stackrel{\text{def}}{=} \alpha \quad (\sigma \times \tau)^{\underline{S}} \stackrel{\text{def}}{=} \sigma^{\underline{S}} \times \tau^{\underline{S}} \quad 1^{\underline{S}} \stackrel{\text{def}}{=} \underline{1} \quad (\sigma \multimap \tau)^{\underline{S}} \stackrel{\text{def}}{=} !(\sigma^{\underline{S}}) \otimes \underline{S} \multimap !(\tau^{\underline{S}}) \otimes \underline{S}$$

We extend this translation to type contexts, taking an FGCBV type context Γ to an ECBV type context $\Gamma^{\underline{S}}$.

The translation on terms is syntax-directed. We pick a variable s , completely fresh. The translation takes an FGCBV value type judgement $\Gamma \vdash^v V : \sigma$ to an ECBV judgement $\Gamma^{\underline{S}} \mid - \vdash V^{\underline{S}} : \sigma^{\underline{S}}$, and an FGCBV producer judgement $\Gamma \vdash^p M : \sigma$ to an ECBV judgement $\Gamma^{\underline{S}} \mid s : \underline{S} \vdash M_s^{\underline{S}} : !(\sigma^{\underline{S}}) \otimes \underline{S}$. The translation is defined as follows.

$$\begin{aligned} x^{\underline{S}} &\stackrel{\text{def}}{=} x & \star^{\underline{S}} &\stackrel{\text{def}}{=} \star & \langle V, W \rangle^{\underline{S}} &\stackrel{\text{def}}{=} \langle V^{\underline{S}}, W^{\underline{S}} \rangle & (\pi_1(V))^{\underline{S}} &\stackrel{\text{def}}{=} \pi_1(V^{\underline{S}}) & (\pi_2(V))^{\underline{S}} &\stackrel{\text{def}}{=} \pi_2(V^{\underline{S}}) \\ (\mathbf{return} V)_s^{\underline{S}} &\stackrel{\text{def}}{=} ! (V^{\underline{S}}) \otimes s & (M \mathbf{to} x. N)_s^{\underline{S}} &\stackrel{\text{def}}{=} M_s^{\underline{S}} \mathbf{to} (!x \otimes s). N_s^{\underline{S}} \\ (\lambda x. N)^{\underline{S}} &\stackrel{\text{def}}{=} \lambda z. z \mathbf{to} (!x \otimes s). N_s^{\underline{S}} & (V W)_s^{\underline{S}} &\stackrel{\text{def}}{=} V^{\underline{S}} [!(W^{\underline{S}}) \otimes s] \end{aligned}$$

In the case for λ -abstraction, the z is chosen to be completely fresh.

The translation respects types. For instance

$$\frac{\Gamma, x : \sigma \vdash^p N : \tau}{\Gamma \vdash^v \lambda x. N : \sigma \multimap \tau} \quad \text{becomes} \quad \frac{\frac{\Gamma^{\underline{S}} \mid z : !\sigma^{\underline{S}} \otimes \underline{S} \vdash z : !\sigma^{\underline{S}} \otimes \underline{S} \quad \Gamma^{\underline{S}}, x : \sigma^{\underline{S}} \mid s : \underline{S} \vdash N_s^{\underline{S}} : !\tau^{\underline{S}} \otimes \underline{S}}{\Gamma^{\underline{S}} \mid z : !\sigma^{\underline{S}} \otimes \underline{S} \vdash z \mathbf{to} (!x \otimes s). N_s^{\underline{S}} : !\tau^{\underline{S}} \otimes \underline{S}}}{\Gamma^{\underline{S}} \mid - \vdash \lambda z. z \mathbf{to} (!x \otimes s). N_s^{\underline{S}} : !\sigma^{\underline{S}} \otimes \underline{S} \multimap !\tau^{\underline{S}} \otimes \underline{S}}$$

Theorem 4.1. *The linear-use state-passing translation is sound:*

- (1) *If $\Gamma \vdash^v V \equiv W : \sigma$ then $\Gamma^{\underline{S}} \mid - \vdash V^{\underline{S}} \equiv W^{\underline{S}} : \sigma^{\underline{S}}$.*
- (2) *If $\Gamma \vdash^p M \equiv N : \sigma$ then $\Gamma^{\underline{S}} \mid s : \underline{S} \vdash M_s^{\underline{S}} \equiv N_s^{\underline{S}} : \sigma^{\underline{S}}$.*

This result can be proved by induction on the structure of equality (\equiv) derivations, but it can also be derived semantically as we shall see in Section 5.4.

4.1. Full completeness. We now state our main theorems: fullness on types and full completeness on terms. To state fullness on types we need to talk about isomorphism of types in ECBV. This can be defined in the usual way: for value types, an isomorphism $\mathbf{A} \cong \mathbf{B}$ is given by two judgements, $x : \mathbf{A} \mid - \vdash t : \mathbf{B}$ and $y : \mathbf{B} \mid - \vdash u : \mathbf{A}$, such that $u[t/y] \equiv x$ and $t[u/x] \equiv y$. For computation types, $\underline{\mathbf{A}} \cong \underline{\mathbf{B}}$ is witnessed by closed terms of type $\underline{\mathbf{A}} \multimap \underline{\mathbf{B}}$, $\underline{\mathbf{B}} \multimap \underline{\mathbf{A}}$ composing in both directions to identities. We note the following type isomorphisms, inherited from the enriched effect calculus [8, §3]:

$$\underline{\mathbf{A}} \cong \underline{!} \otimes \underline{\mathbf{A}} \quad \underline{\mathbf{A}} \otimes (\underline{!} \mathbf{B} \otimes \underline{\mathbf{C}}) \cong \underline{!}(\mathbf{A} \times \mathbf{B}) \otimes \underline{\mathbf{C}} \quad (4.1)$$

Theorem 4.2 (Fullness on types). *Let \mathbf{A} be a value type of ECBV formed using no other computation type constants than $\underline{\mathbf{S}}$. Then there exists an FGCBV type σ such that $\sigma^{\underline{\mathbf{S}}} \cong \mathbf{A}$.*

Proof. By induction on the structure of types. The interesting case $\underline{\mathbf{A}} \multimap \underline{\mathbf{B}}$ uses the fact that any computation type not using any $\underline{\alpha}$ other than $\underline{\mathbf{S}}$ is isomorphic to one of the form $!C \otimes \underline{\mathbf{S}}$, which follows from the isomorphisms (4.1). \square

We now state our main syntactic result.

Theorem 4.3 (Full completeness).

- (1) *If $\Gamma \vdash^v V, W : \sigma$ and $\Gamma^{\underline{\mathbf{S}}} \vdash V^{\underline{\mathbf{S}}} \equiv W^{\underline{\mathbf{S}}} : \sigma^{\underline{\mathbf{S}}}$ then $\Gamma \vdash^v V \equiv W : \sigma$.*
- (2) *If $\Gamma \vdash^p M, N : \sigma$ and $\Gamma^{\underline{\mathbf{S}}} \mid s : \underline{\mathbf{S}} \vdash M_s^{\underline{\mathbf{S}}} \equiv N_s^{\underline{\mathbf{S}}} : !\sigma^{\underline{\mathbf{S}}} \otimes \underline{\mathbf{S}}$ then $\Gamma \vdash^p M \equiv N : \sigma$.*
- (3) *For any $\Gamma^{\underline{\mathbf{S}}} \mid - \vdash t : \sigma^{\underline{\mathbf{S}}}$ there is a term $\Gamma \vdash^v V : \sigma$ such that $\Gamma^{\underline{\mathbf{S}}} \mid - \vdash t \equiv V^{\underline{\mathbf{S}}} : \sigma^{\underline{\mathbf{S}}}$.*
- (4) *For any $\Gamma^{\underline{\mathbf{S}}} \mid s : \underline{\mathbf{S}} \vdash t : !(\sigma^{\underline{\mathbf{S}}}) \otimes \underline{\mathbf{S}}$ there is a producer term $\Gamma \vdash^p M : \sigma$ such that $\Gamma^{\underline{\mathbf{S}}} \mid s : \underline{\mathbf{S}} \vdash t \equiv M_s^{\underline{\mathbf{S}}} : !(\sigma^{\underline{\mathbf{S}}}) \otimes \underline{\mathbf{S}}$.*

In Section 5.4 we sketch a semantic proof of Theorems 4.2 and 4.3.

5. A SEMANTIC PROOF OF FULL COMPLETENESS

In this section we present two constructions on models. The first (§5.1) constructs a Kleisli model (Def. 3.1) from an enriched model (Def. 2.1) with a specified state object. The second (§5.2) constructs an enriched model with a state object from a given Kleisli model. The state-passing translation arises from the first of these constructions. These two constructions form a bi-adjunction exhibiting the category of Kleisli models as a coreflective subcategory of the category of enriched models with chosen state objects (§5.3). In Section 5.4 we shall see how to use these facts to explain full completeness of the linear-use state-passing translation (Theorem 4.3).

5.1. From enriched models with state to Kleisli models. Given an enriched call-by-value model (\mathbf{V}, \mathbf{C}) with a state object $\underline{\mathbf{S}}$ in \mathbf{C} , we can form an enriched Kleisli model $\mathbf{Kl}(\mathbf{V}, \mathbf{C}, \underline{\mathbf{S}}) \stackrel{\text{def}}{=} (\mathbf{V}, \mathbf{Kl}_{\underline{\mathbf{S}}}, J_{\underline{\mathbf{S}}})$, where the category $\mathbf{Kl}_{\underline{\mathbf{S}}}$ has the same objects as \mathbf{V} and hom-sets

$$\text{Hom}_{\mathbf{Kl}_{\underline{\mathbf{S}}}}(A, B) \stackrel{\text{def}}{=} \text{Hom}_{\mathbf{C}}(A \cdot \underline{\mathbf{S}}, B \cdot \underline{\mathbf{S}})$$

Composition in $\mathbf{Kl}_{\underline{\mathbf{S}}}$ is just composition as in \mathbf{C} . (This is an isomorphic presentation of the Kleisli category for the monad $\mathbf{C}(\underline{\mathbf{S}}, - \cdot \underline{\mathbf{S}})$ on \mathbf{V} .) The functor $J_{\underline{\mathbf{S}}}$ is the identity on objects and maps $f : A \rightarrow B$ to $f \cdot \underline{\mathbf{S}}$.

Lemma 5.1. *For any enriched call-by-value model with state $(\mathbf{V}, \mathbf{C}, \underline{\mathbf{S}})$ the data $(\mathbf{V}, \mathbf{Kl}_{\underline{\mathbf{S}}}, J_{\underline{\mathbf{S}}})$ defines an enriched Kleisli model.*

Proof. The action $(-)_1 \cdot_{\mathbf{Kl}} (-)_2 : \mathbf{V} \times \mathbf{Kl}_{\underline{\mathbf{S}}} \rightarrow \mathbf{Kl}_{\underline{\mathbf{S}}}$ is defined on objects as $A \cdot_{\mathbf{Kl}} B = A \times B$. On morphisms it maps $f : A \rightarrow A', g : B \cdot \underline{\mathbf{S}} \rightarrow B' \cdot \underline{\mathbf{S}}$ to the composite

$$(A \times B) \cdot \underline{\mathbf{S}} \xrightarrow{\cong} A \cdot (B \cdot \underline{\mathbf{S}}) \xrightarrow{f \cdot g} A' \cdot (B' \cdot \underline{\mathbf{S}}) \xrightarrow{\cong} (A' \times B') \cdot \underline{\mathbf{S}}$$

which is easily seen to be functorial.

The right adjoint to $(-)_1 \cdot_{\mathbf{Kl}} A$ is $\mathbf{Kl}_{\underline{\mathbf{S}}}(A, -) \stackrel{\text{def}}{=} \mathbf{C}(A \cdot \underline{\mathbf{S}}, (-) \cdot \underline{\mathbf{S}})$. \square

The construction \mathbf{Kl} described above extends to a 2-functor $\mathbf{Kl}: \mathcal{Ecbv} \rightarrow \mathcal{Kleisli}$ from the 2-category of enriched models to the 2-category of Kleisli models. See Appendix C.2 for details.

5.2. From Kleisli models to enriched models with state. Any Kleisli model is trivially an enriched model, so for the opposite construction we just need to pick a state object in a Kleisli model. We define $\mathbf{St}(\mathbf{V}, \mathbf{C}, J) \stackrel{\text{def}}{=} (\mathbf{V}, \mathbf{C}, 1)$, where 1 is the terminal object of \mathbf{V} considered as an object of \mathbf{C} . This definition extends to a 2-functor $\mathbf{St}: \mathcal{Kleisli} \rightarrow \mathcal{Ecbv}$, as shown in Appendix C.

The motivation for this definition is that, as we now show, the 2-category \mathcal{Ecbv} can be seen as a 2-category of enriched adjunctions, and the 2-functor \mathbf{St} can be seen as an inclusion of Kleisli adjunctions into \mathcal{Ecbv} .

Let (\mathbf{V}, \mathbf{C}) be an enriched model. By an *enriched adjunction* we mean an adjunction $F \dashv U: \mathbf{C} \rightarrow \mathbf{V}$ equipped with a natural coherent isomorphism $F(A \times B) \cong A \cdot F(B)$. When \mathbf{V} is cartesian closed, this is equivalent to the usual definition, i.e. a natural isomorphism $\mathbf{C}(F(-)_1, -)_2 \cong \mathbf{V}(-)_1, U(-)_2)$ in \mathbf{V} (see e.g. [20]).

Any choice of state object gives an enriched adjunction, since $(- \cdot \underline{S})$ is left adjoint to $\mathbf{C}(\underline{S}, -): \mathbf{C} \rightarrow \mathbf{V}$. The following proposition (first noted for EEC, [8, Proof of Thm. 4], [7]) shows that every enriched adjunction arises in this way:

Proposition 5.2 ([7]). *Let (\mathbf{V}, \mathbf{C}) be an enriched model. If $F \dashv U: \mathbf{C} \rightarrow \mathbf{V}$ is an enriched adjunction then it is naturally isomorphic to the enriched adjunction induced by $F(1)$.*

So enriched adjunctions correspond essentially bijectively to state objects. In particular the state object corresponding to a Kleisli model is 1. Monads induced by state objects can be described in ECBV as $\underline{S} \multimap !(-) \otimes \underline{S}$. By the correspondence between Kleisli models and strong monads we arrive at the slogan: *Every strong monad is a linear-use state monad*. More directly, the slogan can be derived from the isomorphism $\mathbf{Kl}_T(1, A \times 1) \cong T(A)$, which holds for the Kleisli category \mathbf{Kl}_T of any strong monad T .

(*Remark:* if T is a strong monad on \mathbf{V} then, for any object \underline{S} of \mathbf{Kl}_T , the linear-use state monad $\mathbf{Kl}_T(\underline{S}, (-) \cdot \underline{S})$ on \mathbf{V} is also known as the application of the state monad transformer to T , as in [28].)

5.3. A coreflection. The constructions \mathbf{Kl} and \mathbf{St} form a bi-adjunction between the 2-categories of Kleisli models and of enriched models with state. One intuition for this is that it expresses the minimality property of Kleisli resolutions of monads.

Theorem 5.3. *The 2-functor $\mathbf{St}: \mathcal{Kleisli} \rightarrow \mathcal{Ecbv}$ is left biadjoint to \mathbf{Kl} , i.e., for any pair of objects $(\mathbf{V}, \mathbf{C}, J)$ and $(\mathbf{V}', \mathbf{C}', \underline{S})$ of $\mathcal{Kleisli}$ and \mathcal{Ecbv} respectively, there is an equivalence of categories*

$$\mathcal{Ecbv}(\mathbf{St}(\mathbf{V}, \mathbf{C}, J), (\mathbf{V}', \mathbf{C}', \underline{S})) \simeq \mathcal{Kleisli}((\mathbf{V}, \mathbf{C}, J), \mathbf{Kl}(\mathbf{V}', \mathbf{C}', \underline{S}))$$

natural in $(\mathbf{V}, \mathbf{C}, J)$ and $(\mathbf{V}', \mathbf{C}', \underline{S})$. Moreover, the unit of the adjunction $\eta: id_{\mathcal{Ecbv}} \rightarrow \mathbf{Kl} \circ \mathbf{St}$ is an isomorphism.

See Appendix C for a proof of Theorem 5.3. Since left bi-adjoints preserve bi-initial objects we get the following connection between the syntactic enriched model $(\mathcal{V}_{\text{ECBV}}, \mathcal{C}_{\text{ECBV}}, \underline{S})$ and the syntactic Kleisli model $(\mathcal{V}_{\text{FGCBV}}, \mathcal{C}_{\text{FGCBV}}, J)$.

Corollary 5.4. $(\mathcal{V}_{\text{ECBV}}, \mathcal{C}_{\text{ECBV}}, \underline{\mathbb{S}})$ and $\mathbf{St}(\mathcal{V}_{\text{FGCBV}}, \mathcal{C}_{\text{FGCBV}}, J)$ are equivalent as objects of \mathfrak{Ecbv} .

Since $\mathbf{Kl}(\mathbf{St}(\mathbf{V}, \mathbf{C}, J)) = (\mathbf{V}, \mathbf{Kl}_1, J_1)$ the unit of the adjunction can be described as the pair $(id_{\mathbf{V}}, G: \mathbf{C} \rightarrow \mathbf{Kl}_1)$ where $G(f: A \rightarrow B)$ is the composite

$$A \cdot 1 \xrightarrow{\cong} A \xrightarrow{f} B \xrightarrow{\cong} B \cdot 1$$

using the isomorphism $J(\pi): A \cdot 1 = A \times 1 \rightarrow A$. The pair $(id_{\mathbf{V}}, G)$ preserves the enrichment only up to isomorphism, and this is our main motivation for using 2-categories of models (see also the discussion in Section 2.5).

5.4. A semantic explanation of the state-passing translation. The linear-use state-passing translation is essentially the interpretation of fine-grain call-by-value into the model obtained by applying the construction \mathbf{Kl} of Section 5 to the syntactic enriched call-by-value model of Section 2.4. In this model judgements $\Gamma \vdash^v V: \sigma$ and $\Gamma \vdash^p M: \sigma$ are interpreted as judgements of the form

$$x: \prod \Gamma^{\underline{\mathbb{S}}} \mid - \vdash \llbracket V \rrbracket: \sigma^{\underline{\mathbb{S}}} \qquad - \mid x: !(\prod \Gamma^{\underline{\mathbb{S}}}) \otimes \underline{\mathbb{S}} \vdash \llbracket M \rrbracket: \sigma^{\underline{\mathbb{S}}}$$

respectively, where $\prod \Gamma^{\underline{\mathbb{S}}}$ is the product of all the types appearing in $\Gamma^{\underline{\mathbb{S}}}$.

Lemma 5.5. *Let $(\mathcal{V}_{\text{ECBV}}, \mathcal{C}_{\text{ECBV}}, \underline{\mathbb{S}})$ be the syntactic enriched Kleisli model of Section 2.4. The interpretation of FGCBV into $\mathbf{Kl}(\mathcal{V}_{\text{ECBV}}, \mathcal{C}_{\text{ECBV}}, \underline{\mathbb{S}})$ models a type σ as $\sigma^{\underline{\mathbb{S}}}$. Let $\Gamma = x_1: \sigma_1 \dots x_n: \sigma_n$ be a context of FGCBV and let $\Gamma \vdash^v V: \tau$ and $\Gamma \vdash^p M: \tau$ be judgements of FGCBV. Then V and M are interpreted as the equivalence classes of the terms*

$$\begin{aligned} x: (\prod \Gamma^{\underline{\mathbb{S}}}) \mid - \vdash V^{\underline{\mathbb{S}}}[\pi_1 x \dots \pi_n x / x_1 \dots x_n]: \tau^{\underline{\mathbb{S}}} \\ - \mid x: !(\prod \Gamma^{\underline{\mathbb{S}}}) \otimes \underline{\mathbb{S}} \vdash x \text{ to } (!z \otimes s). (M^{\underline{\mathbb{S}}}[\pi_1 z \dots \pi_n z / x_1 \dots x_n]): \tau^{\underline{\mathbb{S}}} \end{aligned}$$

Soundness of the state-passing translation (Theorem 4.1) follows immediately from Lemma 5.5. Fullness on types and full completeness (Theorems 4.2 and 4.3) are also consequences.

Proof of Theorems 4.2 and 4.3. By Theorem 3.3 and Lemma 5.5 the state-passing translation is the unique (up to isomorphism) 1-cell

$$(F, G): (\mathcal{V}_{\text{FGCBV}}, \mathcal{C}_{\text{FGCBV}}, J) \rightarrow \mathbf{Kl}(\mathcal{V}_{\text{ECBV}}, \mathcal{C}_{\text{ECBV}}, \underline{\mathbb{S}})$$

in $\mathfrak{Kleisli}$. It suffices to show that this is an equivalence in $\mathfrak{Kleisli}$, because this implies that F and G are both equivalences of categories, in particular they are essentially full on objects (proving Theorem 4.2) and full and faithful (proving Theorem 4.3).

By initiality (F, G) must be isomorphic to the composite

$$(\mathcal{V}_{\text{FGCBV}}, \mathcal{C}_{\text{FGCBV}}, J) \xrightarrow{\eta} \mathbf{Kl}(\mathbf{St}(\mathcal{V}_{\text{FGCBV}}, \mathcal{C}_{\text{FGCBV}}, J)) \xrightarrow{\cong} \mathbf{Kl}(\mathcal{V}_{\text{ECBV}}, \mathcal{C}_{\text{ECBV}}, \underline{\mathbb{S}})$$

of the unit of the adjunction (which is an isomorphism by Theorem 5.3) and \mathbf{Kl} applied to the equivalence of Corollary 5.4. Since this composite is an equivalence, so is (F, G) . \square

6. SUMS

It is routine to add sum types to the languages considered in Section 2 and 3, and the state-passing translation extends straightforwardly. We now summarize the details.

6.1. Sums in the enriched call-by-value calculus. We add sums to the enriched call-by-value calculus, for both value and computation types. The required modifications to the calculus are given in Figure 3. The resulting calculus is still a fragment of the enriched effect calculus [7]. We now extend the notion of model to accommodate the sums.

Terminology. Recall that a *distributive category* is a category with finite products and coproducts such that the canonical morphisms $0 \rightarrow A \times 0$ and $((A \times B) + (A \times C)) \rightarrow (A \times (B + C))$ are isomorphisms.

If a distributive category \mathbf{V} has an action (\cdot) on a category \mathbf{C} with finite coproducts $(0, \oplus)$, then we say that the situation is *distributive* if the four canonical morphisms $0 \rightarrow A \cdot 0$, $(A \cdot B) \oplus (A \cdot C) \rightarrow A \cdot (B \oplus C)$, $0 \rightarrow 0 \cdot A$ and $(A \cdot C) \oplus (B \cdot C) \rightarrow (A + B) \cdot C$ are isomorphisms. If the action is enriched, i.e. each functor $(- \cdot A) : \mathbf{V} \rightarrow \mathbf{C}$ has a right adjoint, then this definition of distributive coproducts amounts to the usual notion of enriched coproducts. (Note that when the action is enriched then the third and fourth canonical morphisms are necessarily isomorphisms since left adjoints preserve colimits.)

Definition 6.1. A *distributive enriched model* is given by a distributive category \mathbf{V} and a category \mathbf{C} enriched in \mathbf{V} with copowers and enriched coproducts.

It is straightforward to extend the sound interpretation of the enriched call-by-value calculus in enriched models (§2.2) to a sound interpretation of enriched call-by-value calculus with sums in distributive enriched models.

Of the examples in Section 2.3, (1)–(5) are distributive. The syntactic model of the version of the calculus with sums is a distributive enriched model, and it is bi-initial for the suitable generalization of morphism.

6.2. Sums in the fine-grain call-by-value calculus. It is equally straightforward to add sums to the fine-grain call-by-value language. This is summarized in Figure 4.

We only include constructors/destructors as value terms, but from these we can derive constructors/destructors for producer terms, as follows.

$$\begin{aligned} \text{image}^p(M) &\stackrel{\text{def}}{=} M \text{ to } x. \text{return }?(x) & \text{in}_i^p(M) &\stackrel{\text{def}}{=} M \text{ to } x. \text{return in}_i(x) \\ \text{case}^p M \text{ of } (\text{in}_1(x_1).N_1 | \text{in}_2(x_2).N_2) & & & \\ &\stackrel{\text{def}}{=} M \text{ to } z. (\text{case } z \text{ of } (\text{in}_1(x_1).\lambda w. N_1 | \text{in}_2(x_2).\lambda w. N_2))(\star) \end{aligned}$$

where w, z are fresh.

These constructions have derived typing rules

$$\frac{\Gamma \vdash^p M : 0}{\Gamma \vdash^p \text{image}^p(M) : \sigma} \quad \frac{\Gamma \vdash^p M_i : \sigma_i}{\Gamma \vdash^p \text{in}_i^p(M) : \sigma_1 + \sigma_2} \quad (i = 1, 2) \tag{6.1}$$

$$\frac{\Gamma \vdash^p M : \sigma_1 + \sigma_2 \quad \Gamma, x_i : \sigma_i \vdash^p N_i : \tau \quad (i = 1, 2)}{\Gamma \vdash^p \text{case}^p M \text{ of } (\text{in}_1(x_1).N_1 | \text{in}_2(x_2).N_2) : \tau}$$

Types.

$$\begin{aligned} \mathbf{A}, \mathbf{B} &::= \alpha \mid \underline{1} \mid \mathbf{A} \times \mathbf{B} \mid \underline{\mathbf{A}} \multimap \underline{\mathbf{B}} \mid 0 \mid \mathbf{A} + \mathbf{B} \\ \underline{\mathbf{A}}, \underline{\mathbf{B}} &::= \underline{\alpha} \mid !\mathbf{A} \otimes \underline{\mathbf{B}} \mid \underline{0} \mid \underline{\mathbf{A}} \oplus \underline{\mathbf{B}} . \end{aligned}$$

Term formation. The following rules are in addition to the rules in Figure 1.

$$\begin{aligned} &\frac{\Gamma \mid - \vdash t : 0}{\Gamma \mid - \vdash ?(t) : \mathbf{A}} \quad \frac{\Gamma \mid \Delta \vdash t : \underline{0}}{\Gamma \mid \Delta \vdash \underline{?}(t) : \underline{\mathbf{A}}} \\ &\frac{\Gamma \mid - \vdash t : \mathbf{A}_i}{\Gamma \mid - \vdash \text{in}_i(t) : \mathbf{A}_1 + \mathbf{A}_2} \quad (i = 1, 2) \quad \frac{\Gamma \mid \Delta \vdash t : \underline{\mathbf{A}}_i}{\Gamma \mid \Delta \vdash \underline{\text{in}}_i(t) : \underline{\mathbf{A}}_1 \oplus \underline{\mathbf{A}}_2} \quad (i = 1, 2) \\ &\frac{\Gamma \mid - \vdash t : \mathbf{A}_1 + \mathbf{A}_2 \quad \Gamma, x_1 : \mathbf{A}_1 \mid - \vdash u_1 : \mathbf{C} \quad \Gamma, x_2 : \mathbf{A}_2 \mid - \vdash u_2 : \mathbf{C}}{\Gamma \mid - \vdash \text{case } t \text{ of } (\text{in}_1(x_1). u_1 \mid \text{in}_2(x_2). u_2) : \mathbf{C}} \\ &\frac{\Gamma \mid \Delta \vdash s : \underline{\mathbf{A}}_1 \oplus \underline{\mathbf{A}}_2 \quad \Gamma \mid x_1 : \underline{\mathbf{A}}_1 \vdash t_1 : \underline{\mathbf{C}} \quad \Gamma \mid x_2 : \underline{\mathbf{A}}_2 \vdash t_2 : \underline{\mathbf{C}}}{\Gamma \mid \Delta \vdash \underline{\text{case}} s \text{ of } (\underline{\text{in}}_1(x_1). t_1 \mid \underline{\text{in}}_2(x_2). t_2) : \underline{\mathbf{C}}} \end{aligned}$$

Equality. The following rules are in addition to the rules in Figure 1.

$$\begin{aligned} &\frac{\Gamma \mid - \vdash t : 0 \quad \Gamma, x : 0 \mid - \vdash u : \mathbf{A}}{\Gamma \mid - \vdash ?(t) \equiv u[t/x] : \mathbf{A}} \quad \frac{\Gamma \mid \Delta \vdash t : \underline{0} \quad \Gamma \mid x : \underline{0} \vdash u : \underline{\mathbf{A}}}{\Gamma \mid \Delta \vdash \underline{?}(t) \equiv u[t/x] : \underline{\mathbf{A}}} \\ &\frac{\Gamma \mid - \vdash t : \mathbf{A}_i \quad \Gamma, x_1 : \mathbf{A}_1 \mid - \vdash u_1 : \mathbf{B} \quad \Gamma, x_2 : \mathbf{A}_2 \mid - \vdash u_2 : \mathbf{B}}{\Gamma \mid - \vdash \text{case } (\text{in}_i(t)) \text{ of } (\text{in}_1(x_1). u_1 \mid \text{in}_2(x_2). u_2) \equiv u_i[t/x_i] : \mathbf{B}} \quad (i = 1, 2) \\ &\frac{\Gamma \mid \Delta \vdash t : \underline{\mathbf{A}}_i \quad \Gamma \mid x_1 : \underline{\mathbf{A}}_1 \vdash u_1 : \underline{\mathbf{B}} \quad \Gamma \mid x_2 : \underline{\mathbf{A}}_2 \vdash u_2 : \underline{\mathbf{B}}}{\Gamma \mid \Delta \vdash \underline{\text{case}} (\underline{\text{in}}_i(t)) \text{ of } (\underline{\text{in}}_1(x_1). u_1 \mid \underline{\text{in}}_2(x_2). u_2) \equiv u_i[t/x_i] : \underline{\mathbf{B}}} \quad (i = 1, 2) \\ &\frac{\Gamma \mid - \vdash t : \mathbf{A}_1 + \mathbf{A}_2 \quad \Gamma, z : \mathbf{A}_1 + \mathbf{A}_2 \mid - \vdash u : \mathbf{B}}{\Gamma \mid - \vdash u[t/z] \equiv \text{case } t \text{ of } (\text{in}_1(x_1). u[\text{in}_1(x_1)/z] \mid \text{in}_2(x_2). u[\text{in}_2(x_2)/z]) : \mathbf{B}} \\ &\frac{\Gamma \mid \Delta \vdash t : \underline{\mathbf{A}}_1 \oplus \underline{\mathbf{A}}_2 \quad \Gamma \mid z : \underline{\mathbf{A}}_1 \oplus \underline{\mathbf{A}}_2 \vdash u : \underline{\mathbf{B}}}{\Gamma \mid \Delta \vdash u[t/z] \equiv \underline{\text{case}} t \text{ of } (\underline{\text{in}}_1(x_1). u[\text{in}_1(x_1)/z] \mid \underline{\text{in}}_2(x_2). u[\text{in}_2(x_2)/z]) : \underline{\mathbf{B}}} \end{aligned}$$

Figure 3: Additional rules for sum types in Enriched Call-by-Value

<i>Types.</i>	
$\sigma ::= \alpha \mid 1 \mid \sigma \times \sigma \mid \sigma \multimap \sigma \mid 0 \mid \sigma + \sigma$	
<hr/>	
<i>Term formation.</i> The following rules are in addition to the rules in Figure 2.	
$\frac{\Gamma \vdash^v V : 0}{\Gamma \vdash^v ?(V) : \sigma}$	$\frac{\Gamma \vdash^v V : \sigma_1}{\Gamma \vdash^v \mathbf{in}_1(V) : \sigma_1 + \sigma_2}$
$\frac{\Gamma \vdash^v V : \sigma_2}{\Gamma \vdash^v \mathbf{in}_2(V) : \sigma_1 + \sigma_2}$	
$\frac{\Gamma \vdash^v V : \sigma_1 + \sigma_2 \quad \Gamma, x_1 : \sigma_1 \vdash^v W_1 : \tau \quad \Gamma, x_1 : \sigma_2 \vdash^v W_2 : \tau}{\Gamma \vdash^v \mathbf{case} V \text{ of } (\mathbf{in}_1(x_1).W_1 \mid \mathbf{in}_2(x_2).W_2) : \tau}$	
<hr/>	
<i>Equality.</i> The following rules are in addition to the rules in Figure 2.	
$\frac{\Gamma \vdash^v V : 0 \quad \Gamma, x : 0 \vdash^v W : \sigma}{\Gamma \vdash^v ?(V) \equiv W[V/x] : \sigma}$	
$\frac{\Gamma \vdash^v V : \sigma_i \quad \Gamma, x_1 : \sigma_1 \vdash^v W_1 : \tau \quad \Gamma, x_1 : \sigma_2 \vdash^v W_2 : \tau}{\Gamma \vdash^v \mathbf{case} \mathbf{in}_i(V) \text{ of } (\mathbf{in}_1(x_1).W_1 \mid \mathbf{in}_2(x_2).W_2) \equiv W_i[V/x_i] : \tau} \quad (i = 1, 2)$	
$\frac{\Gamma \vdash^v V : \sigma_1 + \sigma_2 \quad \Gamma, z : \sigma_1 + \sigma_2 \vdash^v W : \tau}{\Gamma \vdash^v W[V/z] \equiv \mathbf{case} V \text{ of } (\mathbf{in}_1(x_1).W[\mathbf{in}_1(x_1)/z] \mid \mathbf{in}_2(x_2).W[\mathbf{in}_2(x_2)/z]) : \tau}$	

Figure 4: Additional rules for sum types in Fine-Grain Call-by-Value.

For example:

$$\frac{\frac{\frac{\Gamma, x_1 : \sigma_1 \vdash^p N_1 : \tau}{\Gamma, x_1 : \sigma_1 \vdash^v \lambda w. N_1 : 1 \multimap \tau} \quad \frac{\Gamma, x_2 : \sigma_2 \vdash^p N_2 : \tau}{\Gamma, x_2 : \sigma_2 \vdash^v \lambda w. N_2 : 1 \multimap \tau}}{\Gamma, z \vdash^v \mathbf{case} z \text{ of } (\mathbf{in}_1(x_1).\lambda w. N_1 \mid \mathbf{in}_2(x_2).\lambda w. N_2) : 1 \multimap \tau} \quad \Gamma, z \vdash^v \star : 1}{\Gamma \vdash^p M : \sigma_1 + \sigma_2 \quad \Gamma, z : \sigma_1 + \sigma_2 \vdash^p (\mathbf{case} z \text{ of } (\mathbf{in}_1(x_1).\lambda w. N_1 \mid \mathbf{in}_2(x_2).\lambda w. N_2)) (\star) : \tau}}{\Gamma \vdash^p \mathbf{case}^p M \text{ of } (\mathbf{in}_1(x_1).N_1 \mid \mathbf{in}_2(x_2).N_2) : \tau}$$

We now refine the notion of Kleisli model (Def. 3.1) to accommodate the sums.

Definition 6.2. A *distributive enriched Kleisli model* (distributive Kleisli model for short) is a distributive enriched model (Def 6.1) together with an identity-on-objects functor $J : \mathbf{V} \rightarrow \mathbf{C}$ that strictly preserves copowers and coproducts.

Note that in any Kleisli model, J will preserve coproducts because it has a right adjoint, $\mathbf{C}(1, -)$. We insist moreover that it preserves coproducts strictly.

Note that a distributive Kleisli model is what Power [41, Def. 36] calls a distributive closed Freyd category.

It is straightforward to extend our interpretation of the fine-grain call-by-value calculus in Kleisli models (§3.1) to an interpretation of the calculus with sums in distributive Kleisli models. The interpretation is sound and there is a syntactic model.

In Section 3.2 we discussed the equivalence between enriched Kleisli models and strong monads with Kleisli exponentials. This equivalence extends to an equivalence between distributive enriched Kleisli models, and strong monads on distributive categories with Kleisli exponentials. Likewise, the constructions **St** and **Kl** of Section 5 extend to constructions deriving distributive enriched models from distributive Kleisli models and vice versa, and an extension of Theorem 5.3 states that **St** and **Kl** exhibit a 2-category of distributive Kleisli models as a coreflective subcategory of a 2-category of distributive enriched models.

6.3. Sums and the state-passing translation. It is straightforward to adapt the state-passing translation to accommodate sums. Recall that each type σ of FGCBV is translated to a value type $\sigma^{\mathbb{S}}$ of ECBV. We set

$$0^{\mathbb{S}} \stackrel{\text{def}}{=} 0 \quad (\sigma + \tau)^{\mathbb{S}} \stackrel{\text{def}}{=} \sigma^{\mathbb{S}} + \tau^{\mathbb{S}}.$$

We extend this translation to type contexts, taking an FGCBV type context Γ to an ECBV type context $\Gamma^{\mathbb{S}}$.

Recall that the translation on terms takes an FGCBV value type judgement $\Gamma \vdash^v V : \sigma$ to an ECBV judgement $\Gamma^{\mathbb{S}} \mid - \vdash V^{\mathbb{S}} : \sigma^{\mathbb{S}}$, and takes an FGCBV producer judgement $\Gamma \vdash^p M : \sigma$ to an ECBV judgement $\Gamma^{\mathbb{S}} \mid s : \underline{\mathbb{S}} \vdash M_s^{\mathbb{S}} : !(\sigma^{\mathbb{S}}) \otimes \underline{\mathbb{S}}$. We extend the translation in Section 4 straightforwardly, as follows:

$$\begin{aligned} ?(V)^{\mathbb{S}} &\stackrel{\text{def}}{=} ?(V^{\mathbb{S}}) & \text{in}_1(V)^{\mathbb{S}} &\stackrel{\text{def}}{=} \text{in}_1(V^{\mathbb{S}}) & \text{in}_2(V)^{\mathbb{S}} &\stackrel{\text{def}}{=} \text{in}_2(V^{\mathbb{S}}) \\ \text{case } V \text{ of } (\text{in}_1(x_1).W_1 \mid \text{in}_2(x_2).W_2)^{\mathbb{S}} &\stackrel{\text{def}}{=} \text{case } V^{\mathbb{S}} \text{ of } (\text{in}_1(x_1).W_1^{\mathbb{S}} \mid \text{in}_2(x_2).W_2^{\mathbb{S}}) \end{aligned}$$

The translation remains sound. The full definability results (Theorems 4.2 and 4.3) continue to hold in the presence of sums.

7. REMARKS ON THE LINEAR-USE CONTINUATION-PASSING TRANSLATION

We now briefly emphasise that the linear-use continuation-passing translation arises as a formal dual of the linear-use state-passing translation. This is not a new observation: Hasegawa noticed it in the context of classical linear logic ([15, §8],[33]) and indeed it informed the earlier work on the enriched effect calculus.

The linear-use continuation-passing translation was first elaborated by Berdine, O’Hearn, Reddy and Thielecke [4], but our main reference is the more recent work by Egger, Møgelberg and Simpson [9, 10]. They showed that the linear-use continuation-passing translation can be extended to an involutive translation of the enriched effect calculus to itself, and derived a full completeness result from this. That work, in turn, stems from Hasegawa’s full completeness result [15] for a linear-use continuation-passing translation into dual intuitionistic / linear logic.

Following [9], our development is fuelled by the following categorical observation. If a category \mathbf{C} is enriched in \mathbf{V} with copowers, then we can form the dual category \mathbf{C}^{op} which is also enriched in \mathbf{V} , but now with powers instead of copowers. (Recall that an enriched category \mathbf{C} has powers \underline{Y}^X if the functor $\text{Hom}_{\mathbf{V}}(X, \mathbf{C}(-, \underline{Y})) : \mathbf{C}^{\text{op}} \rightarrow \mathbf{Set}$ is

Types.

$$\mathbf{A}, \mathbf{B} ::= \alpha \mid 1 \mid \mathbf{A} \times \mathbf{B} \mid \mathbf{A} \multimap \mathbf{B}$$

$$\underline{\mathbf{A}}, \underline{\mathbf{B}} ::= \underline{\alpha} \mid \mathbf{A} \rightarrow \underline{\mathbf{B}} .$$

Term formation.

$$\frac{}{\Gamma, x: \mathbf{A}, \Gamma' \mid - \vdash x: \mathbf{A}} \quad \frac{}{\Gamma \mid z: \underline{\mathbf{A}} \vdash z: \underline{\mathbf{A}}} \quad \frac{}{\Gamma \mid - \vdash \star: 1}$$

$$\frac{\Gamma \mid - \vdash t: \mathbf{A} \quad \Gamma \mid - \vdash u: \mathbf{B}}{\Gamma \mid - \vdash \langle t, u \rangle: \mathbf{A} \times \mathbf{B}} \quad \frac{\Gamma \mid - \vdash t: \mathbf{A}_1 \times \mathbf{A}_2}{\Gamma \mid - \vdash \pi_i(t): \mathbf{A}_i}$$

$$\frac{\Gamma \mid z: \underline{\mathbf{A}} \vdash t: \underline{\mathbf{B}}}{\Gamma \mid - \vdash \underline{\lambda} z. t: \underline{\mathbf{A}} \multimap \underline{\mathbf{B}}} \quad \frac{\Gamma \mid - \vdash s: \underline{\mathbf{A}} \multimap \underline{\mathbf{B}} \quad \Gamma \mid \Delta \vdash t: \underline{\mathbf{A}}}{\Gamma \mid \Delta \vdash s[t]: \underline{\mathbf{B}}}$$

$$\frac{\Gamma, x: \mathbf{A} \mid \Delta \vdash t: \underline{\mathbf{B}}}{\Gamma \mid \Delta \vdash \lambda x. t: \mathbf{A} \rightarrow \underline{\mathbf{B}}} \quad \frac{\Gamma \mid \Delta \vdash t: \mathbf{A} \rightarrow \underline{\mathbf{B}} \quad \Gamma, x: \mathbf{A} \mid - \vdash u: \mathbf{A}}{\Gamma \mid \Delta \vdash t u: \underline{\mathbf{B}}}$$

Equations: Equations of Figure 1 but with equations for tensor types replaced by:

$$\frac{\Gamma, x: \mathbf{A} \mid \Delta \vdash t: \underline{\mathbf{B}} \quad \Gamma \mid - \vdash u: \mathbf{A}}{\Gamma \mid \Delta \vdash (\lambda x. t) u \equiv t[u/x]: \underline{\mathbf{B}}} \quad \frac{\Gamma \mid \Delta \vdash t: \mathbf{A} \rightarrow \underline{\mathbf{B}}}{\Gamma, x: \mathbf{A} \mid \Delta \vdash t \equiv \lambda x. (t x): \mathbf{A} \rightarrow \underline{\mathbf{B}}}$$

Figure 5: A CPS variant of the enriched call-by-value calculus.

representable.) When viewed under this duality, the state-passing translation becomes the continuation-passing translation, as we now explain.

In Figure 5, we provide an internal language for enriched categories with powers. We call this the ‘CPS variant of ECBV’, because it is the target of the continuation passing translation (see below). The key difference with Figure 1 is that we have replaced the tensor type $(!A \otimes B)$ by a power type $(A \rightarrow B)$. It is another fragment of the enriched effect calculus. If \mathbf{V} is a category with products and \mathbf{C} is a category enriched in \mathbf{V} with powers, then we can interpret this CPS variant of ECBV in (\mathbf{V}, \mathbf{C}) through a variation of the interpretation in Section 2.2. The power type is interpreted using the categorical powers: $[A \rightarrow B] \stackrel{\text{def}}{=} [B][A]$. A computation judgement $\Gamma \mid \Delta \vdash t: \underline{A}$ is interpreted as a morphism $[[t]] : [[\Delta]] \rightarrow [[A]]^{[\Gamma]}$ in \mathbf{C} .

Following the categorical analysis above, we define a bijection $(-)^{\circ}$ between the types of ECBV with this CPS variant:

$$\begin{aligned} \alpha^{\circ} &\stackrel{\text{def}}{=} \alpha & 1^{\circ} &\stackrel{\text{def}}{=} 1 & (\mathbf{A} \times \mathbf{B})^{\circ} &\stackrel{\text{def}}{=} (\mathbf{A}^{\circ} \times \mathbf{B}^{\circ}) \\ (\underline{\mathbf{A}} \multimap \underline{\mathbf{B}})^{\circ} &\stackrel{\text{def}}{=} (\underline{\mathbf{B}}^{\circ} \multimap \underline{\mathbf{A}}^{\circ}) & (!\mathbf{A} \otimes \underline{\mathbf{B}})^{\circ} &\stackrel{\text{def}}{=} \mathbf{A}^{\circ} \rightarrow \underline{\mathbf{B}}^{\circ} \end{aligned} \tag{7.1}$$

This bijection extends to terms-in-context straightforwardly, and the resulting translation is a restriction of the linear-use cps involution of the enriched effect calculus studied in [9, 10].

We achieve a linear-use continuation-passing translation by composing the state-passing translation of Section 4 with this bijection (7.1). For clarity, we now write this out explicitly. We fix a computation type \underline{R} of ECBV, thought of as a return type. We translate FGCBV types σ to ECBV value types $\sigma^{\underline{R}}$:

$$\alpha^{\underline{R}} \stackrel{\text{def}}{=} \alpha \quad (\sigma \times \tau)^{\underline{R}} \stackrel{\text{def}}{=} \sigma^{\underline{R}} \times \tau^{\underline{R}} \quad 1^{\underline{R}} \stackrel{\text{def}}{=} 1 \quad (\sigma \multimap \tau)^{\underline{R}} \stackrel{\text{def}}{=} ((\tau^{\underline{R}}) \rightarrow \underline{R}) \multimap ((\sigma^{\underline{R}}) \rightarrow \underline{R})$$

We extend this translation to type contexts, taking an FGCBV type context Γ to an ECBV type context $\Gamma^{\underline{R}}$.

The translation on terms is syntax-directed. We pick a variable k , completely fresh. The translation takes an FGCBV value type judgement $\Gamma \vdash^v V : \sigma$ to an ECBV judgement $\Gamma^{\underline{R}} \mid - \vdash V^{\underline{R}} : \sigma^{\underline{R}}$, and it takes an FGCBV producer judgement $\Gamma \vdash^p M : \sigma$ to an ECBV judgement $\Gamma^{\underline{R}} \mid k : \sigma^{\underline{R}} \rightarrow \underline{R} \vdash M_k^{\underline{R}} : \underline{R}$.

$$\begin{aligned} x^{\underline{R}} &\stackrel{\text{def}}{=} x & \star^{\underline{R}} &\stackrel{\text{def}}{=} \star & \langle V, W \rangle^{\underline{R}} &\stackrel{\text{def}}{=} \langle V^{\underline{R}}, W^{\underline{R}} \rangle & (\pi_1(V))^{\underline{R}} &\stackrel{\text{def}}{=} \pi_1(V^{\underline{R}}) & (\pi_2(V))^{\underline{R}} &\stackrel{\text{def}}{=} \pi_2(V^{\underline{R}}) \\ (\text{return } V)_k^{\underline{R}} &\stackrel{\text{def}}{=} k(V^{\underline{R}}) & (M \text{ to } x. N)_k^{\underline{R}} &\stackrel{\text{def}}{=} (\lambda k. M_k^{\underline{R}})[\lambda x. N_k^{\underline{R}}] \\ (\lambda x. N)^{\underline{R}} &\stackrel{\text{def}}{=} \lambda k. \lambda x. N_k^{\underline{R}} & (V W)_k^{\underline{R}} &\stackrel{\text{def}}{=} (V^{\underline{R}}[k]) W^{\underline{R}} \end{aligned}$$

The continuation-passing translation inherits the following results from the soundness and full completeness of the state-passing translation (Theorems 4.1 and 4.3).

Proposition 7.1. *For any computation type \underline{R} , the continuation-passing translation is sound:*

- (1) *If $\Gamma \vdash^v V \equiv W : \sigma$ then $\Gamma^{\underline{R}} \mid - \vdash V^{\underline{R}} \equiv W^{\underline{R}} : \sigma^{\underline{R}}$.*
- (2) *If $\Gamma \vdash^p M \equiv N : \sigma$ then $\Gamma^{\underline{R}} \mid k : \sigma^{\underline{R}} \rightarrow \underline{R} \vdash M_k^{\underline{R}} \equiv N_k^{\underline{R}} : \underline{R}$.*

Proposition 7.2 ([9], Corollary 1). *Let \underline{R} be a computation type constant. The continuation-passing translation is fully complete, in the following sense.*

- (1) *If $\Gamma \vdash^v V, W : \sigma$ and $\Gamma^{\underline{R}} \mid - \vdash V^{\underline{R}} \equiv W^{\underline{R}} : \sigma^{\underline{R}}$ then $\Gamma \vdash^v V \equiv W : \sigma$.*
- (2) *If $\Gamma \vdash^p M, N : \sigma$ and $\Gamma^{\underline{R}} \mid k : \sigma^{\underline{R}} \rightarrow \underline{R} \vdash M_k^{\underline{R}} \equiv N_k^{\underline{R}} : \underline{R}$ then $\Gamma \vdash^p M \equiv N : \sigma$.*
- (3) *For any $\Gamma^{\underline{R}} \mid - \vdash t : \sigma^{\underline{R}}$ there is $\Gamma \vdash^v V : \sigma$ such that $\Gamma^{\underline{R}} \mid - \vdash t \equiv V^{\underline{R}} : \sigma^{\underline{R}}$.*
- (4) *For any $\Gamma^{\underline{R}} \mid k : \sigma^{\underline{R}} \rightarrow \underline{R} \vdash t : \underline{R}$ there is a producer term $\Gamma \vdash^p M : \sigma$ such that $\Gamma^{\underline{R}} \mid k : \sigma^{\underline{R}} \rightarrow \underline{R} \vdash t \equiv M_k^{\underline{R}} : \underline{R}$.*

The full completeness result is the same as the one obtained by Egger et al. [9, Corollary 1] except that loc. cit. describes a translation on the full enriched effect calculus rather than this fragment of it.

7.1. Sums and products. The CPS variant of the enriched call-by-value calculus can be extended so that value types are closed under sums and computation types are closed under products. Thus the types are as follows:

$$\begin{aligned} \mathbf{A}, \mathbf{B} &::= \alpha \mid 1 \mid \mathbf{A} \times \mathbf{B} \mid \underline{\mathbf{A}} \multimap \underline{\mathbf{B}} \mid 0 \mid \mathbf{A} + \mathbf{B} \\ \underline{\mathbf{A}}, \underline{\mathbf{B}} &::= \underline{\alpha} \mid \underline{\mathbf{A}} \rightarrow \underline{\mathbf{B}} \mid \underline{1} \mid \underline{\mathbf{A}} \times \underline{\mathbf{B}} . \end{aligned}$$

(For brevity, we omit the term language, which is a fragment of the enriched effect calculus.) The type system is designed to be dual to the enriched call-by-value language with sums. The translation from that language to this one (7.1) is extended as follows:

$$(0)^\circ \stackrel{\text{def}}{=} 0 \quad (A + B)^\circ \stackrel{\text{def}}{=} A^\circ + B^\circ \quad (0)^\circ \stackrel{\text{def}}{=} \underline{1} \quad (A \oplus B)^\circ \stackrel{\text{def}}{=} A^\circ \times B^\circ$$

and the analysis of Section 6.3 can be converted to a fully-complete linear-use continuation-passing style translation from fine-grain call-by-value with sums.

8. EFFECT THEORIES

To illustrate the nature of the state-passing translation we endow our calculi with effects. We do this in a general way, by following the programme of Plotkin, Power and others [38] whereby a theory of effects is presented as an algebraic theory.

We discuss how to add effects to the source and target languages of the state-passing translation, FGCBV and ECBV. Our central observation is that to accommodate an algebraic theory of effects in the enriched call-by-value calculus it is necessary and sufficient to supply the chosen state type \underline{S} with the structure of a comodel. The idea of state being a comodel, particularly for the theory of store, arose in the work of Plotkin, Power and Shkaravska [40, 35].

The section is structured as follows. We begin with an overview in which we study the situation for a particular algebraic effect. We then proceed to look at a general notion of effect theory (§8.3), its relationship with the state-passing translation (§8.4) and notions of model and comodel (§8.5,8.6).

8.1. Overview. We give an overview of the situation in terms of a particular algebraic theory: an algebraic theory for accessing a single bit of memory. This is an algebraic theory in the classical sense (like monoids, groups, rings, etc.). It has one binary operation ($?$), a unary operation (f) and the following four equations:

$$(v ? x) ? (y ? z) \equiv v ? z \quad x \equiv x ? x \quad f(f(x)) \equiv x \quad f(x ? y) \equiv f(y) ? f(x) \quad (8.1)$$

Here is some intuition. If x and y are computations, then $x ? y$ is the computation that first reads the bit in memory and then branches to x or to y depending on whether the bit was set. If x is a computation then $f(x)$ is the computation that first flips the bit in memory (0 to 1, 1 to 0) and then continues as x . There are derived operations $\text{update}_0(x) \stackrel{\text{def}}{=} x ? f(x)$ and $\text{update}_1(x) \stackrel{\text{def}}{=} f(x) ? x$, which first write 0 (resp. 1) and then continue as x .

We now explain how to accommodate this algebraic theory in the fine-grain and enriched call-by-value calculi.

8.1.1. Fine-grain call-by-value: algebraic operations and generic effects. In the fine-grain call-by-value calculus (§3), the algebraic theory (8.1) can be accommodated in two equivalent ways: by adding algebraic operations and by adding generic effects. In adding the operations, we add the following term formation rules for each type σ :

$$\frac{\Gamma \vdash^p M : \sigma \quad \Gamma \vdash^p N : \sigma}{\Gamma \vdash^p M ?_\sigma N : \sigma} \quad \frac{\Gamma \vdash^p M : \sigma}{\Gamma \vdash^p f_\sigma(M) : \sigma} \quad (8.2)$$

We also add the equations in (8.1) at each type, and an algebraicity equation for each operation (e.g. [43, Def 3.14]):

$$\frac{\Gamma \vdash^p M_1 : \sigma \quad \Gamma \vdash^p M_2 : \sigma \quad \Gamma, x : \sigma \vdash^p N : \tau}{\Gamma \vdash^p (M_1 ?_{\sigma} M_2) \mathbf{to} x. N \equiv (M_1 \mathbf{to} x. N) ?_{\tau} (M_2 \mathbf{to} x. N) : \tau}$$

$$\frac{\Gamma \vdash^p M : \sigma \quad \Gamma, x : \sigma \vdash^p N : \tau}{\Gamma \vdash^p f_{\sigma}(M) \mathbf{to} x. N \equiv f_{\tau}(M \mathbf{to} x. N) : \tau}$$

The result is a programming language with higher types and a single bit of storage.

The second way to accommodate the algebraic theory into the fine-grain call-by-value calculus is by adding generic effects. For this we need sum types (§6.2). The idea is that an expression in n variables in the algebraic theory corresponds to a ground producer term of type n ($= 1 + \dots + 1$). Thus we add the following axioms for term formation:

$$\frac{}{\Gamma \vdash^p \mathbf{deref}(\star) : 2} \qquad \frac{}{\Gamma \vdash^p \mathbf{flip}(\star) : 1} \qquad (8.3)$$

Informally, $\mathbf{deref}(\star)$ is a computation that returns the boolean value of the memory cell, and $\mathbf{flip}(\star)$ is a computation that flips the value in the memory cell. An important observation of Plotkin and Power [38] is that the algebraic operations can be recovered at all types from the generic effects, as follows:

$$M ?_{\sigma} N \stackrel{\text{def}}{=} \mathbf{if}^p \mathbf{deref}(\star) \mathbf{then} M \mathbf{else} N \qquad f_{\sigma}(M) \stackrel{\text{def}}{=} \mathbf{flip}(\star); M$$

where we use some shorthand notation:

$$\mathbf{if}^p \mathbf{deref}(\star) \mathbf{then} M \mathbf{else} N \stackrel{\text{def}}{=} \mathbf{case}^p \mathbf{deref}(\star) \mathbf{of} (\mathbf{in}_1(x_1).M | \mathbf{in}_2(x_2).N)$$

$$\mathbf{flip}(\star); M \stackrel{\text{def}}{=} \mathbf{flip}(\star) \mathbf{to} x. M$$

Conversely the generic effects can be derived from the algebraic operations:

$$\mathbf{deref}(\star) \stackrel{\text{def}}{=} \mathbf{in}_1^p(\star) ?_2 \mathbf{in}_2^p(\star) \qquad \mathbf{flip}(\star) \stackrel{\text{def}}{=} f_1(\star)$$

(The subscript 1 on $f_1(\star)$ is the unit type.) We can thus write the four equations (8.1) directly in terms of generic effects:

$$\vdash^p \mathbf{deref}(\star) \mathbf{to} x. \mathbf{deref}(\star) \mathbf{to} y. \mathbf{return} \langle x, y \rangle \equiv \mathbf{deref}(\star) \mathbf{to} x. \mathbf{return} \langle x, x \rangle : 2 \times 2$$

$$\vdash^p \mathbf{return}(\star) \equiv \mathbf{deref}(\star); \mathbf{return}(\star) : 1 \qquad \vdash^p \mathbf{flip}(\star); \mathbf{flip}(\star) \equiv \mathbf{return}(\star) : 1$$

$$\vdash^p \mathbf{flip}(\star); \mathbf{deref}(\star) \equiv \mathbf{deref}(\star) \mathbf{to} x. \mathbf{flip}(\star); \mathbf{return}(\neg x) : 2 \qquad (8.4)$$

writing $\neg x$ for $\mathbf{if} x \mathbf{then} \mathbf{in}_2(\star) \mathbf{else} \mathbf{in}_1(\star)$.

The two derived operations for writing a bit can be combined into a single command **assign**:

$$\Gamma \vdash^v V : 2$$

$$\frac{}{\Gamma \vdash^p \mathbf{assign}(V) \stackrel{\text{def}}{=} \mathbf{if}^p (\mathbf{deref}(\star) \mathbf{to} x. \mathbf{return} (V \mathbf{xor} x)) \mathbf{then} \mathbf{flip}(\star) \mathbf{else} \mathbf{return}(\star) : 1}$$

where **xor** is the evident binary operation on values of type 2. Using this derived command, the four equations for accessing the bit of memory can be equivalently written as the three

program equations of Plotkin and Power [37]:

$$- \vdash^p \text{return}(\star) \equiv \text{deref}(\star) \text{ to } x. \text{assign}(x):1 \quad (8.5)$$

$$x: 2 \vdash^p \text{assign}(x); \text{deref}(\star) \equiv \text{assign}(x); \text{return}(x):2 \quad (8.6)$$

$$x, y: 2 \vdash^p \text{assign}(x); \text{assign}(y) \equiv \text{assign}(y):1 \quad (8.7)$$

which state that reading a cell and then writing the same value is the same as doing nothing, that writing and the reading yields the value just written, and that the effect of two writes equals that of the second.

8.1.2. *Enriched call-by-value and state access operations.* How can we accommodate the algebraic theory for a bit of memory (8.1) in the enriched call-by-value calculus? In this section we develop the following observation. Whereas in FGCBV each type should be a *model* of the theory, in that (8.2) gives terms (?) and (f) at each type σ , in ECBV the distinguished state type \underline{S} should be a *comodel* of the theory, which means that there are ground value terms

$$\text{read} : \underline{S} \multimap \underline{S} \oplus \underline{S} \quad \text{and} \quad \text{flip} : \underline{S} \multimap \underline{S} \quad (8.8)$$

which we call *state access operations*. (It is called a *comodel* because the arrows have been reversed and (\times) has become (\oplus) .) Using the isomorphism $(\underline{S} \oplus \underline{S}) \cong (!2 \otimes \underline{S})$, we can understand the type of (read) as $\underline{S} \multimap !2 \otimes \underline{S}$. The idea is that the read operation takes a state and returns the value stored in that state. It also returns a state: this is necessary because state is linear and cannot be discarded or duplicated. Notice that, under the state-passing translation, the two generic effects (8.3) become the two state access operations.

The four equations (8.1) are also appealing when written in terms of state access operations.

$$\begin{aligned} - |s: \underline{S} \vdash \text{read}[s] \text{ to } (!b \otimes s'). \text{read}[s'] \text{ to } (!b' \otimes s''). !\langle b, b' \rangle \otimes s'' \\ \equiv \text{read}[s] \text{ to } (!b \otimes s'). !\langle b, b \rangle \otimes s' : !(2 \times 2) \otimes \underline{S} \\ - |s: \underline{S} \vdash s \equiv \text{read}[s] \text{ to } (!b \otimes s'). s' : \underline{S} \end{aligned} \quad (8.9)$$

$$- |s: \underline{S} \vdash \text{flip}[\text{flip}[s]] \equiv s : \underline{S}$$

$$- |s: \underline{S} \vdash \text{read}[\text{flip}[s]] \equiv \text{read}[s] \text{ to } (!b \otimes s'). !\neg b \otimes \text{flip}[s'] : !2 \otimes \underline{S}$$

Notice that the second equation says that the read operation does not change the state.

The two derived operations for writing a bit can be combined into a single state access operation:

$$\begin{aligned} \text{write} \stackrel{\text{def}}{=} \underline{\lambda}x. x \text{ to } (!b \otimes s). \text{read}[s] \text{ to } (!b' \otimes s'). ((\text{if } (b \text{ xor } b') \text{ then } (\underline{\lambda}s. \text{flip}[s]) \text{ else } (\underline{\lambda}s. s))[s']) \\ : !2 \otimes \underline{S} \multimap \underline{S} \end{aligned}$$

Intuitively, $\text{write}[!b \otimes s]$ writes the value b to the state s , returning the updated state.

In Section 4 we have seen a fully-complete state-passing translation from FGCBV to ECBV. This translation extends to a fully-complete translation from FGCBV with generic effects to ECBV with state access operations.

8.1.3. *Continuation passing style and algebraic operations.* Finally, we turn to the linear-use continuation-passing style. In this setting, it is natural to require that the distinguished return type $\underline{\mathbf{R}}$ be a model of the theory. This is dual to the situation with state-passing style, where the distinguished state type $\underline{\mathbf{S}}$ is a comodel of the theory.

More precisely, we extend the CPS variant of ECBV (§ 7) with the theory of memory access by adding ground value terms

$$(?) : \underline{\mathbf{R}} \times \underline{\mathbf{R}} \multimap \underline{\mathbf{R}} \quad \text{and} \quad (f) : \underline{\mathbf{R}} \multimap \underline{\mathbf{R}}$$

satisfying the following equations:

$$\begin{aligned} - |k : (\underline{\mathbf{R}} \times \underline{\mathbf{R}}) \times (\underline{\mathbf{R}} \times \underline{\mathbf{R}}) \vdash ((\pi_1(\pi_1 k)) ? (\pi_1(\pi_2 k))) ? ((\pi_2(\pi_1 k)) ? (\pi_2(\pi_2 k))) \\ \equiv (\pi_1(\pi_1 k)) ? (\pi_2(\pi_2 k)) : \underline{\mathbf{R}} \\ - |k : \underline{\mathbf{R}} \vdash k \equiv k ? k : \underline{\mathbf{R}} \\ - |k : \underline{\mathbf{R}} \vdash f[f[k]] \equiv k : \underline{\mathbf{R}} \\ - |k : \underline{\mathbf{R}} \times \underline{\mathbf{R}} \vdash f[(\pi_1 k) ? (\pi_2 k)] \equiv (f[\pi_2 k]) ? (f[\pi_1 k]) : \underline{\mathbf{R}} \end{aligned}$$

Thus the generic effects in the source language endow the return type $\underline{\mathbf{R}}$ of the linear-use continuation-passing translation with the structure of a model for the algebraic theory.

8.1.4. *Further simple examples of algebraic theories for computational effects.* The theory of accessing a bit of memory is perhaps the simplest example of a stateful effect. The connections between algebraic operations, generic effects and state access operations also work for less state-like effects.

Printing. The algebraic theory of printing a single bit has two unary function symbols, p_0 and p_1 . For instance, the term $p_0(p_1(x))$ should be understood as the computation that first prints 0, then prints 1, then continues as x . There are no equations in this theory.

The generic effects for printing can be grouped together into one producer term

$$x : 2 \vdash^p \mathbf{print}(x) : 1$$

thought of as a command that prints its argument.

As a state access operation, we have a function $\mathbf{print} : !2 \otimes \underline{\mathbf{S}} \multimap \underline{\mathbf{S}}$ which, given a bit and a state, returns a new state. Intuitively, $\underline{\mathbf{S}}$ is a list of everything printed so far, and \mathbf{print} appends its first argument to its second argument.

Probability. There are different algebraic theories of probabilistic choice. The simplest one is the theory of ‘mean-values’ considered by Heckmann [16] (but perhaps first introduced by Aczél [2]): it has one binary function symbol \odot and its axioms are the medial law, idempotence and commutativity:

$$(u \odot x) \odot (y \odot z) \equiv (u \odot y) \odot (x \odot z) \quad x \equiv x \odot x \quad x \odot y \equiv y \odot x$$

The idea is that a computation $x \odot y$ tosses a coin, and proceeds as x if heads and y if tails.

The generic effect for \odot is $\Gamma \vdash^p \mathbf{toss}(\star) : 2$ which, intuitively, tosses the coin and returns the result. In this style, the first equation is written

$$- \vdash^p \mathbf{toss}(\star) \mathbf{to} x. \mathbf{toss}(\star) \mathbf{to} y. (x, y) \equiv \mathbf{toss}(\star) \mathbf{to} y. \mathbf{toss}(\star) \mathbf{to} x. (x, y) : 2 \times 2$$

It says that it doesn’t matter which order you toss coins.

The state access operation $\text{toss}: \underline{S} \multimap !2 \otimes \underline{S}$ can be thought of as making the coin an explicit parameter: we can think of \underline{S} as a type of coins. In this style, the second equation

$$- |s: \underline{S} \vdash s \equiv \text{toss}[s] \text{ to } (!b \otimes s'). s': \underline{S}$$

says that when you toss a coin you get the same coin back. The third equation

$$- |s: \underline{S} \vdash \text{toss}[s] \equiv \text{toss}[s] \text{ to } (!b \otimes s'). (!(-b) \otimes s'): !2 \otimes \underline{S}$$

says that if you toss a coin it is the same as tossing a coin and turning it once more without looking. This illustrates that probability is not really stateful and so for this effect the approach based on algebraic operations is perhaps the most profitable perspective. The point is that different computational effects are better suited to different approaches (algebraic operations, generic effects, and state access operations) even though all three approaches are always available.

8.2. State access operations, algebraic operations, and generic effects. We now make precise the informal connections made between state access operations, generic effects and algebraic operations in the previous section. We do this by revisiting the results of Plotkin and Power [38] in the context of an enriched model (\mathbf{V}, \mathbf{C}) .

In the previous section we focused on the classical situation where arities are natural numbers. However, from the perspective of state access operations, generic effects and algebraic operations have little to do with natural numbers per se. It is just as easy to allow arities to be arbitrary objects of the base category \mathbf{V} . In the following result we do not make an artificial restriction to natural numbers. Instead we consider an operation with arity $[A_1, \dots, A_n]$ and a parameter from B , where A_1, \dots, A_n, B are objects of \mathbf{V} . The classical case of an n -ary operation is recovered by setting the objects A_1, \dots, A_n, B to all be the terminal object 1 .

Theorem 8.1. *Let (\mathbf{V}, \mathbf{C}) be an enriched model with sums. Let \underline{S} be an object of \mathbf{C} . Let $A_1 \dots A_n$ and B be objects of \mathbf{V} . The following data are equivalent:*

- (1) *A state access operation: a morphism $B \cdot \underline{S} \rightarrow A_1 \cdot \underline{S} \oplus \dots \oplus A_n \cdot \underline{S}$ in \mathbf{C} .*
- (2) *A generic effect: a morphism $B \rightarrow T_{\underline{S}}(A_1 + \dots + A_n)$ in \mathbf{V} , where $T_{\underline{S}}$ is the monad $\mathbf{C}(\underline{S}, (-) \cdot \underline{S})$.*
- (3) *An algebraic operation: a \mathbf{V} -natural family of morphisms in \mathbf{V}*

$$\left\{ \prod_{i=1}^n (U_{\underline{S}} \underline{X})^{A_i} \rightarrow (U_{\underline{S}} \underline{X})^B \right\}_{\underline{X} \in \mathbf{C}}$$

where $U_{\underline{S}}(\underline{X}) \stackrel{\text{def}}{=} \mathbf{C}(\underline{S}, \underline{X})$.

The last point requires some explanation. First, even though \mathbf{V} is not cartesian closed, exponentials with base $U_{\underline{S}}(\underline{X})$ exist: $(U_{\underline{S}} \underline{X})^A \cong \mathbf{C}(A \cdot \underline{S}, \underline{X})$. Second, the constructions

$$F \stackrel{\text{def}}{=} \prod_{i=1}^n (U_{\underline{S}}(-))^{A_i} \quad G \stackrel{\text{def}}{=} (U_{\underline{S}}(-))^B$$

can be understood as \mathbf{V} -functors $F, G: \mathbf{C} \rightarrow \mathbf{V}$, since there are families of morphisms

$$\{F_{\underline{X}, \underline{Y}}: \mathbf{C}(\underline{X}, \underline{Y}) \times F(\underline{X}) \rightarrow F(\underline{Y})\}_{\underline{X}, \underline{Y} \in \mathbf{C}} \quad \{G_{\underline{X}, \underline{Y}}: \mathbf{C}(\underline{X}, \underline{Y}) \times G(\underline{X}) \rightarrow G(\underline{Y})\}_{\underline{X}, \underline{Y} \in \mathbf{C}}$$

in \mathbf{V} that satisfy the laws for functors (respecting identities and composition). Thirdly, a family of morphisms $\{\phi_{\underline{X}}: F(\underline{X}) \rightarrow G(\underline{X})\}_{\underline{X} \in \mathbf{C}}$ is called \mathbf{V} -natural if the following diagram

commutes in \mathbf{V} for all \underline{X} and \underline{Y} :

$$\begin{array}{ccc} \mathbf{C}(\underline{X}, \underline{Y}) \times F(\underline{X}) & \xrightarrow{\mathbf{C}(\underline{X}, \underline{Y}) \times \phi_X} & \mathbf{C}(\underline{X}, \underline{Y}) \times G(\underline{X}) \\ F_{\underline{X}, \underline{Y}} \downarrow & & \downarrow G_{\underline{X}, \underline{Y}} \\ F(\underline{Y}) & \xrightarrow{\phi_Y} & G(\underline{Y}) \end{array}$$

(It is perhaps more compelling if algebraic operations are defined as structure on a \mathbf{V} -category of $T_{\underline{S}}$ -algebras, but this \mathbf{V} -category cannot be constructed without further assumptions on \mathbf{V} — see [38, §7].)

Proof of Theorem 8.1. To see the connection between (1) and (2), consider the following bijections:

$$\begin{aligned} \text{Hom}_{\mathbf{C}}(B \cdot \underline{S}, A_1 \cdot \underline{S} \oplus \dots \oplus A_n \cdot \underline{S}) &\cong \text{Hom}_{\mathbf{V}}(B, \mathbf{C}(\underline{S}, A_1 \cdot \underline{S} \oplus \dots \oplus A_n \cdot \underline{S})) \\ &\cong \text{Hom}_{\mathbf{V}}(B, \mathbf{C}(\underline{S}, (A_1 + \dots + A_n) \cdot \underline{S})) \\ &= \text{Hom}_{\mathbf{V}}(B, T_{\underline{S}}(A_1 + \dots + A_n)). \end{aligned}$$

To see the connection between (1) and (3), we note that

$$\prod_{i=1}^n (U_{\underline{S}} \underline{X})^{A_i} \cong \mathbf{C}(A_1 \cdot \underline{S} \oplus \dots \oplus A_n \cdot \underline{S}, \underline{X}) \quad \text{and} \quad (U_{\underline{S}} \underline{X})^B \cong \mathbf{C}(B \cdot \underline{S}, \underline{X})$$

and the enriched Yoneda lemma gives

$$\mathbf{V}\text{-Nat}(\mathbf{C}(A_1 \cdot \underline{S} \oplus \dots \oplus A_n \cdot \underline{S}, -), \mathbf{C}(B \cdot \underline{S}, -)) \cong \text{Hom}_{\mathbf{C}}(B \cdot \underline{S}, A_1 \cdot \underline{S} \oplus \dots \oplus A_n \cdot \underline{S}).$$

We remark that the equivalence of (2) and (3) is essentially Theorem 2 of [38]. \square

8.3. Effect theories. In the previous section we described connection between state access operations, generic effects and algebraic operations. As we explained, the natural level of generality for this is more sophisticated than the classical setting: the arity of an operation is a list and we allow the operation to take a parameter. This suggests a generalization of algebraic theories that we call ‘effect theories’, since they are useful from the computational perspective.

The illustration in Section 8.1 involves storage of a single bit. A motivating example of effect theory arises from modifying that theory above to allow storage of a more interesting datatype. In FGCBV, we would like to have an (abstract) type `Val` of storable values, and generic effects `deref` and `assign` with typing judgements

$$\frac{}{\Gamma \vdash^p \text{deref}(\star) : \text{Val}} \qquad \frac{\Gamma \vdash^v V : \text{Val}}{\Gamma \vdash^p \text{assign}(V) : 1} \tag{8.10}$$

We add to the theory of equality for FGCBV (Fig. 2) the three equations for global store proposed by Plotkin and Power [37] (8.5–(8.7)):

$$\begin{aligned} \vdash^p \text{return}(\star) &\equiv \text{deref}(\star) \text{ to } x. \text{assign}(x) : 1 \\ x : \text{Val} \vdash^p \text{assign}(x); \text{deref}(\star) &\equiv \text{assign}(x); \text{return}(x) : \text{Val} \\ x, y : \text{Val} \vdash^p \text{assign}(x); \text{assign}(y) &\equiv \text{assign}(y) : 1 \end{aligned}$$

Our notion of effect theory accommodates the classical kinds of theory in the overview (§ 8.1) and also the more general kind of theory of memory access illustrated above. It is roughly the same as that used by Plotkin and Pretnar [36, §3]. The main difference is

in the presentation: we use generic effects rather than algebraic operations. Rather than introducing a new calculus for expressing the allowable equations of an effect theory, we use the first-order fragment of FGCBV.

Value theories. Before we introduce effect theories we briefly discuss *value theories*, which are simple extensions of the value judgements of FGCBV. By a value signature we shall simply mean a signature for a many-sorted algebraic theory in the usual sense. This means a set of type constants ranged over by α, β , and a set of term constants f with a given arity $f: (\alpha_1, \dots, \alpha_n) \rightarrow \beta$, where the α_i, β range over type constants. We can extend FGCBV along a value signature by adding the type constants and the typing rule

$$\frac{\Gamma \vdash^v t_i: \alpha_i \ (i = 1, \dots, n)}{\Gamma \vdash^v f(t_1, \dots, t_n): \beta} \quad (8.11)$$

for every term constant $f: (\alpha_1, \dots, \alpha_n) \rightarrow \beta$ in the signature. A value theory is a value signature with a set of equations, i.e. pairs of terms typable in the same context $\Gamma \vdash^v V = W: \beta$, where V, W are formed only using variable introduction and the rule (8.11).

Effect theories. An *effect signature* consists of a value theory and a set of effect constants each with an assigned arity $e: \bar{\beta}; \bar{\alpha}_1 + \dots + \bar{\alpha}_n$ consisting of a list $\bar{\beta}$ of type constants and a formal sum of lists of type constants, $\bar{\alpha}_1 + \dots + \bar{\alpha}_n$. (Here we are abbreviating a list $(\beta_1 \dots \beta_m)$ using the notation $\bar{\beta}$, etc.) FGCBV can be extended along an effect signature by adding, for every effect constant $e: \bar{\beta}; \bar{\alpha}_1 + \dots + \bar{\alpha}_n$, a typing judgement

$$\frac{\Gamma \vdash^v V_1: \beta_1 \quad \dots \quad \Gamma \vdash^v V_m: \beta_m}{\Gamma \vdash^p e(V_1, \dots, V_m): \bar{\alpha}_1 + \dots + \bar{\alpha}_n} \quad (8.12)$$

where $\bar{\beta} = (\beta_1, \dots, \beta_m)$. In the conclusion, the vectors $\bar{\alpha}_i$ should be understood as the product of the types in the vector.

Here are some examples of effect signatures:

- The theory of reading/flipping a bit of memory (§ 8.1) has no value type constants. It has two effect constants, **deref** and **flip**. The effect constant **deref** has arity $1; 1 + 1$ and the effect constant **flip** has arity $1; 1$, where 1 is the empty string.
- The theory for storing an abstract datatype (8.10) has one value type constant **Val** and a pair of effect constants (**deref**: $1; \text{Val}$) and (**assign**: $\text{Val}; 1$). In this case term constants in the value theory can be used to add basic operations manipulating values in **Val**: we could ask that the storable values form a ring. (In future, it would be interesting to allow **Val** to have more structure, for example as an inductive type such as the natural numbers, but it is not clear how to extend the proof of Theorem 10.2 to inductive types.)

An *effect theory* comprises an effect signature and a set of equations. The equations are pairs of producer terms-in-context $\Gamma \vdash^p M \equiv N: \tau$ of a restricted kind: they must be built from the first-order fragment of fine-grain call-by-value in Figure 6. This notion of ‘effect theory’ is connected with the classical notion of algebraic theory in Section 8.1 as follows. If the value theory is empty, with no type constants (α) and no function symbols, then the lists of type constants in (8.12) must all empty, and each generic effect is an operation of arity n . This is the generic effect presentation of an algebraic theory as described in Section 8.1.1.

<p><i>Types.</i></p> $\sigma, \tau ::= \alpha \mid 1 \mid \sigma \times \tau \mid 0 \mid \sigma + \tau$ <hr style="width: 50%; margin: 10px auto;"/> <p><i>Terms.</i> The grammar for terms is as follows. Typing judgements are in Figure 2, Figure 4, and Equations (8.11) and (8.12).</p> $ \begin{aligned} V ::= & x \mid f(V_1, \dots, V_n) \mid \star \mid \pi_1(V) \mid \pi_2(V) \mid \langle V_1, V_2 \rangle \\ & \mid ?(V) \mid \text{in}_1(V) \mid \text{in}_2(V) \mid \text{case } V \text{ of } (\text{in}_1(x_1).W_1 \mid \text{in}_2(x_2).W_2) \\ M ::= & \text{return } V \mid M \text{ to } x. N \mid \text{case}^p M \text{ of } (\text{in}_1(x_1).N_1 \mid \text{in}_2(x_2).N_2) \end{aligned} $

Figure 6: The sub-calculus of fine-grain call-by-value that is used for describing effect theories

8.4. Effect theories and the state-passing translation. An effect theory is an extension to the fine-grain call-by-value language. In Section 4 we explained how the linear-use state-passing translation goes from FGCBV to ECBV. We now explain how ECBV needs to be extended to support this state-passing translation. The restricted nature of the effect theories makes this particularly straightforward and illuminating.

Value theory:

- For each type constant in the value theory, we assume a value type constant.
- For each term constant $f : \bar{\alpha} \rightarrow \beta$ in the value theory we add the following term formation rule:

$$\frac{\Gamma \mid - \vdash t_1 : \alpha_1 \quad \dots \quad \Gamma \mid - \vdash t_n : \alpha_n}{\Gamma \mid - \vdash f(t_1, \dots, t_n) : \beta}$$

- An equation in the value theory must only be formed from variable introduction and the rule (8.11). Thus each equation $\Gamma \vdash^v V \equiv W : \beta$ in the value theory can be understood as an equation $\Gamma \mid - \vdash V \equiv W : \beta$ between value judgements.

Effect signature:

- We assume a chosen computation type constant \underline{S} .
- For each effect constant $e : \bar{\beta}; \bar{\alpha}_1 + \dots + \bar{\alpha}_n$ we add a constant at value type,

$$- \mid - \vdash e : !\bar{\beta} \otimes \underline{S} \multimap !(\bar{\alpha}_1 + \dots + \bar{\alpha}_n) \otimes \underline{S} \tag{8.13}$$

For the theory of reading/flipping a bit, this yields the constants read and flip in (8.8). For the theory of storing an abstract datatype, this yields two state access operations:

$$\text{read} : \underline{S} \multimap !\text{Val} \otimes \underline{S} \qquad \text{write} : !\text{Val} \otimes \underline{S} \multimap \underline{S}$$

State-passing translation:

- Recall that the state-passing translation (§4) takes a producer judgement of FGCBV $\Gamma \vdash^p M : \sigma$ to a computation judgement of ECBV $\Gamma^{\underline{S}} \mid s : \underline{S} \vdash M_s^{\underline{S}} : !\sigma^{\underline{S}} \otimes \underline{S}$. We extend the state-passing translation to operate on effects:

$$(e(V_1 \dots V_m))_s^{\underline{S}} \stackrel{\text{def}}{=} e(!V_1^{\underline{S}}, \dots, V_m^{\underline{S}}) \otimes s$$

- We use this extended state-passing translation to translate the equations in the effect theory into ECBV, in such a way that the extended translation is sound by construction. Each equation in the effect theory $\Gamma \vdash^p M \equiv N : \tau$ becomes an equation in ECBV:

$$\Gamma \mid s : \underline{S} \vdash M_s^{\underline{S}} \equiv N_s^{\underline{S}} : !\tau \otimes \underline{S}.$$

Notice that we do not need to translate the types in Γ because equations in an effect theory must be from the first-order fragment of fine-grain call-by-value (Fig. 6) which is shared with enriched call-by-value. For instance, the equations in the effect theory for reading/flipping a bit (8.4) give rise to the equations on the state object (8.9). The three equations for storing an abstract datatype (8.5–8.7) become the following equations for a state object \underline{S} :

$$\begin{aligned} & - \mid s : \underline{S} \vdash s \equiv \text{write}[\text{read}[s]] : \underline{S} \\ & x : \text{Val} \mid s : \underline{S} \vdash \text{read}[\text{write}![x \otimes s]] \equiv !x \otimes (\text{write}![x \otimes s]) : !\text{Val} \otimes \underline{S} \\ & x, y : \text{Val} \mid s : \underline{S} \vdash \text{write}![y \otimes \text{write}![x \otimes s]] \equiv \text{write}![y \otimes s] : \underline{S} \end{aligned}$$

8.5. Models and comodels of effect theories. Our analysis of effect theories in Sections 8.3 and 8.4 has been syntactic. We now provide a model-theoretic treatment. We define the interpretation of effect theories in Kleisli models (§8.5.2) and enriched models (§8.5.3). We then define what it means to be a model of an effect theory in general terms (§8.5.4).

8.5.1. Models of value theories. Let \mathbf{V} be a distributive category. An interpretation of a value signature in \mathbf{V} is given by interpretations of the type constants α as objects $\llbracket \alpha \rrbracket$ of \mathbf{V} , and interpretations of term constants $f : \bar{\alpha} \rightarrow \beta$ as morphisms $\llbracket f \rrbracket : \llbracket \bar{\alpha} \rrbracket \rightarrow \llbracket \beta \rrbracket$. (Here, if $\bar{\alpha} = (\alpha_1, \dots, \alpha_n)$ then $\llbracket \bar{\alpha} \rrbracket \stackrel{\text{def}}{=} \llbracket \alpha_1 \rrbracket \times \dots \times \llbracket \alpha_n \rrbracket$.) This interpretation is extended to interpret a term in context $\Gamma \vdash^v V : \beta$ as a morphism $\llbracket V \rrbracket : \llbracket \Gamma \rrbracket \rightarrow \llbracket \beta \rrbracket$. An interpretation of a value theory is an interpretation of the signature such that $\llbracket V \rrbracket = \llbracket W \rrbracket$ for each equation $\Gamma \vdash^v V \equiv W : \beta$ in the value theory.

8.5.2. Interpreting effect theories in Kleisli models. Let $(\mathbf{V}, \mathbf{C}, J)$ be a distributive Kleisli model (Def. 6.2) and suppose an interpretation of the type constants α, β is given. An interpretation of an effect theory E in $(\mathbf{V}, \mathbf{C}, J)$ is given by an interpretation of the value theory in \mathbf{V} and an interpretation of each effect constant $e : \bar{\beta}; \bar{\alpha}_1 + \dots + \bar{\alpha}_n$ in E as a morphism $\llbracket e \rrbracket : \llbracket \bar{\beta} \rrbracket \rightarrow (\llbracket \bar{\alpha}_1 \rrbracket + \dots + \llbracket \bar{\alpha}_n \rrbracket)$ in \mathbf{C} , satisfying the equations of the theory.

8.5.3. Comodels of effect theories in enriched models. Let (\mathbf{V}, \mathbf{C}) be a distributive enriched model in the sense of Definition 6.1. Thus \mathbf{V} is a distributive category and \mathbf{C} is a category enriched in \mathbf{V} with copowers and coproducts. A *comodel* of the effect theory in \mathbf{C} is an object \underline{S} of \mathbf{C} together with a morphism $\llbracket e \rrbracket : \llbracket \bar{\beta} \rrbracket \cdot \underline{S} \rightarrow (\llbracket \bar{\alpha}_1 \rrbracket + \dots + \llbracket \bar{\alpha}_n \rrbracket) \cdot \underline{S}$ in \mathbf{C} for every effect constant $e : \bar{\beta}; \bar{\alpha}_1 + \dots + \bar{\alpha}_n$ such that for each equation $\Gamma \vdash^p M \equiv N : \tau$ in the effect theory, the interpretations of M and N under the state passing style yield equal morphisms:

$$\llbracket M_s^{\underline{S}} \rrbracket = \llbracket N_s^{\underline{S}} \rrbracket : \llbracket \Gamma \rrbracket \cdot \underline{S} \rightarrow \llbracket \tau \rrbracket \cdot \underline{S}.$$

8.5.4. *Models of effect theories in dual enriched models.* We now justify our use of the term ‘comodel’ in Section 8.5.3 by showing that it is a generalization of the standard usage, i.e., dual to the concept of model of an algebraic theory known from classical algebra. Further investigations of the notion of effect theory used in this paper along with relations to existing notions of enriched algebraic theories [23, 39] could be an interesting topic of further research.

To dualize the notion of comodel, let $(\mathbf{V}, \mathbf{C}^{\text{op}})$ be a distributive enriched model, i.e., let \mathbf{V} be a distributive category and \mathbf{C} a category enriched in \mathbf{V} with powers and products, as in Section 7. A *model* of the effect theory in \mathbf{C} is a comodel in \mathbf{C}^{op} . Explicitly this amounts to an object \underline{R} of \mathbf{C} together with a morphism $\llbracket e \rrbracket : \underline{R}^{\llbracket \bar{\alpha}_1 \rrbracket + \dots + \llbracket \bar{\alpha}_n \rrbracket} \rightarrow \underline{R}^{\llbracket \bar{\beta} \rrbracket}$ between powers in \mathbf{C} for every effect constant $e : \bar{\beta}; \bar{\alpha}_1 + \dots + \bar{\alpha}_n$ such that for each equation $\Gamma \vdash^p M \equiv N : \tau$ in the effect theory, the interpretations of M and N in continuation passing style yield equivalent morphisms:

$$\llbracket M_k^{\underline{R}} \rrbracket, \llbracket N_k^{\underline{R}} \rrbracket : \underline{R}^{\llbracket \tau \rrbracket} \rightarrow \underline{R}^{\llbracket \Gamma \rrbracket}.$$

Because the terms of the effect theory are of a restricted kind, it is straightforward to directly describe the interpretations of effect terms as morphisms between powers of \underline{R} , by induction on the structure of typing derivations. For instance, consider the **case^p** rule in (6.1). Given interpretations $\llbracket M_k^{\underline{R}} \rrbracket : \underline{R}^{\llbracket \sigma_1 + \sigma_2 \rrbracket} \rightarrow \underline{R}^{\llbracket \Gamma \rrbracket}$ and $\llbracket (N_i)_k^{\underline{R}} \rrbracket : \underline{R}^{\llbracket \tau \rrbracket} \rightarrow \underline{R}^{\llbracket \Gamma, \sigma_i \rrbracket}$ ($i = 1, 2$), the interpretation $\llbracket (\text{case}^p M \text{ of } (\text{in}_1(x_1).N_1 | \text{in}_2(x_2).N_2))_k^{\underline{R}} \rrbracket$ is the composite

$$\underline{R}^{\llbracket \tau \rrbracket} \xrightarrow{\llbracket (N_1)_k^{\underline{R}}, (N_2)_k^{\underline{R}} \rrbracket} \underline{R}^{\llbracket \Gamma, \sigma_1 \rrbracket} \times \underline{R}^{\llbracket \Gamma, \sigma_2 \rrbracket} \cong \underline{R}^{\llbracket (\sigma_1 + \sigma_2) \rrbracket \times \llbracket \Gamma \rrbracket} \xrightarrow{\llbracket M_k^{\underline{R}} \rrbracket^{\llbracket \Gamma \rrbracket}} \underline{R}^{\llbracket \Gamma \rrbracket \times \llbracket \Gamma \rrbracket} \xrightarrow{\underline{R}^\Delta} \underline{R}^{\llbracket \Gamma \rrbracket}.$$

As another example, $\llbracket (\text{return } V)_k^{\underline{R}} \rrbracket \stackrel{\text{def}}{=} \underline{R}^{\llbracket V \rrbracket} : \underline{R}^{\llbracket \tau \rrbracket} \rightarrow \underline{R}^{\llbracket \Gamma \rrbracket}$ if $\Gamma \vdash^v V : \tau$.

We now return to the setting of classical algebra, when the value theory has no type constants or value constants. We will show that models in the above sense are models of algebraic theories in the classical sense. If there are no type constants then every type in the effect language (Figure 6) is isomorphic to one of the form $1 + 1 + \dots + 1$. The arity of an effect constant $e : \bar{\beta}; \bar{\alpha}_1 + \dots + \bar{\alpha}_n$ must comprise $\bar{\beta}$ as the empty list (since there are no type constants) and $\bar{\alpha}_1 + \dots + \bar{\alpha}_n$ must be a sequence of n empty lists. Thus the interpretation of e in the model is a morphism $\llbracket e \rrbracket : \underline{R}^n \rightarrow \underline{R}$.

We now explain the nature of equations in this restricted setting. In what follows, we will write a natural number n for the type that is the n -fold sum of 1. We will write $\text{in}_i(\star)$ for the i th injection of type n , where $1 \leq i \leq n$, and we will make use of n -ary case constructions, **case** V **of** $(\text{in}_1(\star).W_1 | \dots | \text{in}_n(\star).W_n)$ to destruct terms V of type n . These are just syntactic shorthand for terms that can be defined in the language in Figure 6.

We can focus on equations where Γ is empty. This is because every context has a finite number of ground valuations — if a variable x in a context has type n , then it could be valued with $\text{in}_1(\star) \dots \text{in}_n(\star)$ — and, moreover, an effect equation $\Gamma \vdash^p M \equiv N : n$ is satisfied if and only if it is satisfied at each ground instantiation.

The next step is to note that every effect term $- \vdash^p M : 1 + 1 \dots + 1$ is equal to one built from the following rules:

$$\frac{- \vdash_n^p M_1 : n \quad \dots \quad - \vdash_n^p M_m : n}{- \vdash_n^p \text{case}^p e(\star) \text{ of } (\text{in}_1(\star).M_1 | \dots | \text{in}_m(\star).M_m) : n} (e : -; m)$$

$$\frac{}{- \vdash_n^p \text{in}_i^p(\text{return } \star) : n} \quad 1 \leq i \leq n$$

It is informative to look at the interpretation of these normalized terms. Given interpretations $\llbracket M_1 \rrbracket, \dots, \llbracket M_m \rrbracket : \underline{R}^n \rightarrow \underline{R}$, we have

$$\begin{aligned} \llbracket \text{case}^p e(\star) \text{ of } (\text{in}_1(\star).M_1 \mid \dots \mid \text{in}_m(\star).M_m) \rrbracket &= \underline{R}^n \xrightarrow{(\llbracket M_1 \rrbracket, \dots, \llbracket M_m \rrbracket)} \underline{R}^m \xrightarrow{\llbracket e \rrbracket} \underline{R}, \\ \llbracket \text{in}_i^p(\text{return } \star) \rrbracket &= \underline{R}^n \xrightarrow{\pi_i} \underline{R}. \end{aligned}$$

Thus we see that, in the situation where there are no type constants or value constants, the new general notion of model is the classical notion of model for an algebraic theory.

8.6. Examples of set-theoretic models and comodels. We revisit the simple example effect theories from Sections 8.1 and 8.3 from the model-theoretic perspective. In each case, we find that there are comodels that are state-like.

8.6.1. Storage. The category **Set** is enriched in itself with copowers given by products and the enrichment given by the function space. The set $2 = \{0, 1\}$ is a comodel for the theory of accessing a bit of memory (§8.1), with $\text{read}(x) = (x, x)$ and $\text{flip}(x) = \neg x$. This is a comodel for the theory in the enriched model **(Set, Set)**. Power and Shkaravska [40] showed that 2 is the final comodel in **Set** for the theory of accessing a bit of memory.

As an aside, we note that **Set** is actually equivalent to the category of models for the theory of accessing a bit of memory. The theory of *reading* a bit of memory is sometimes called the theory of ‘rectangular bands’ because every model is isomorphic to one of the form $X \times Y$, with $(x, y) \cdot (x', y') \stackrel{\text{def}}{=} (x, y')$. The anti-involution operation (f) enforces that the model is isomorphic to one of the form $X \times X$, and thus determined by a single set. This phenomenon has been investigated in a more general setting by Métayer [30] and Mesablishvili [29].

We can consider set theoretic models of the theory of storing an abstract datatype (8.10). What is needed is an interpretation Val of the value sort, which also plays the role of the state object. We let $\text{read}(x) = (x, x)$ and $\text{write}(v, s) = v$. This is a comodel for the theory for store in the enriched model **(Set, Set)**.

In both cases, the induced monad on **Set** is the store monad $((-) \times \underline{S})^{\underline{S}}$.

8.6.2. Printing. Let 2^*-Act be the category of algebras for the theory of printing a bit. The objects are triples (X, p_0, p_1) where $p_0, p_1 : X \rightarrow X$, and the morphisms $(X, p_{X,0}, p_{X,1}) \rightarrow (Y, p_{Y,0}, p_{Y,1})$ are functions $X \rightarrow Y$ that commute with the operations. As any ordinary category, this category is enriched in **Set**. It has copowers given by

$$A \cdot (X, p_{X,0}, p_{X,1}) \stackrel{\text{def}}{=} (A \times X, p_{(A \cdot X),0}, p_{(A \cdot X),1}) \quad \text{where } p_{(A \cdot X),i}(a, x) \stackrel{\text{def}}{=} (a, p_{X,i}(x)).$$

Thus **(Set, 2^*-Act)** is an enriched model.

The algebra structure of each algebra equips it with the structure of a comodel in the category of algebras. The leading example is the set 2^* of strings over $\{0, 1\}$, with $p_{2^*,i}(s) = si$. The induced state monad $2^*\text{-Act}(2^*, (-) \cdot 2^*)$ is isomorphic to the monad $2^* \times (-)$ on **Set**. We can understand a string in 2^* as a state: it is the list of things output so far.

8.6.3. *Probability.* Let \mathbf{MValg} be the category of mean-value algebras. The objects are pairs (X, \odot) of a set X and a binary operation \odot satisfying the laws of mean-value algebras (§8.1.4). The pair $(\mathbf{Set}, \mathbf{MValg})$ is an enriched model.

The one-element set is trivially a mean-value algebra, and it can be given the structure of a comodel in the category of mean-value algebras. We can understand the one-element set as a set of states: this captures the idea that probability is a stateless notion of computation. Nonetheless, this ‘state object’ induces a ‘state monad’ on \mathbf{Set} . This can be understood as a monad D of finite dyadic probability distributions. By a finite dyadic probability distribution on a set X , we mean a function $p : X \rightarrow [0, 1]$ such that $\text{supp}(p) = \{x \in X \mid p(x) \neq 0\}$ is finite, $\sum_{x \in \text{supp}(p)} p(x) = 1$, and for all x , $p(x)$ has a finite binary representation. The monad D has $D(X)$ as the set of all finite dyadic probability distributions; the unit picks out the Kronecker distributions, and multiplication $\mu_X : D(D(X)) \rightarrow D(X)$ takes a distribution $p : D(X) \rightarrow [0, 1]$ on $D(X)$ to a distribution $\mu_X(p) : X \rightarrow [0, 1]$ on X , given by $\mu_X(p)(x) \stackrel{\text{def}}{=} \sum_{q \in \text{supp}(p)} (p(q) \times q(x))$.

In general, when the state object is a terminal object then the induced monad preserves terminal objects. A terminal-object-preserving monad is sometimes called affine [25, Thm. 2.1] and the corresponding effects are said to be discardable (e.g. [12], [47, Def. 4.2.4]) since the following rule is admissible in the fine-grain call-by-value language.

$$\frac{\Gamma \vdash^p t : A \quad \Gamma \vdash^p u : B}{\Gamma \vdash^p t \text{ to } x. u \equiv u : B} \quad (x \text{ not free in } u)$$

8.7. **Relating notions of (co)model for effect theories.** We now extend Theorem 8.1 to show, for each effect theory E , a bijective correspondence between the following: comodel structures on \underline{S} , interpretations of E in the Kleisli model $(\mathbf{V}, \mathbf{Kl}_S, J_S)$, algebraic operations equipping each $U_{\underline{S}} X$ with a model structure for E . The latter notion requires some explanation, because the definition of model given in Section 8.5.4 defines only what it means for \underline{R} in a category \mathbf{C} to be a model of E if $(\mathbf{V}, \mathbf{C}^{\text{op}})$ is an enriched model, and in the setting of Theorem 8.1 \mathbf{V}^{op} is generally not \mathbf{V} -enriched.

To generalize the notion of model, let \mathbf{V} be a distributive category, let R be a fixed object of \mathbf{V} such that all exponents of the form R^A exist (i.e. $\mathbf{V}(- \times A, R)$ is representable for all A) and let an interpretation of the value theory of E be given. We define a model structure for E on R to be an interpretation of E in the Kleisli model $(\mathbf{V}, \mathbf{Kl}_{R^{R(-)}}, J)$ where $\mathbf{Kl}_{R^{R(-)}}$ has the same objects as \mathbf{V} , but where a morphism $A \rightarrow B$ in $\mathbf{Kl}_{R^{R(-)}}$ is a morphism $R^B \rightarrow R^A$ in \mathbf{V} . This is isomorphic to the Kleisli category for the strong monad $R^{R(-)}$ on \mathbf{V} . By construction, the model structure interprets each effect constant $e : \bar{\beta}; \bar{\alpha}_1 + \dots + \bar{\alpha}_n$ as a morphism

$$[[e]] : R^{([\bar{\alpha}_1] + \dots + [\bar{\alpha}_n])} \rightarrow R^{[\bar{\beta}]}$$

If \mathbf{V} is cartesian closed then $(\mathbf{V}, \mathbf{V}^{\text{op}})$ is an enriched model and the above definition of a model structure for E on R is equivalent to the one given in Section 8.5.4.

Theorem 8.2. *Let (\mathbf{V}, \mathbf{C}) be an enriched model with sums, let \underline{S} be an object of \mathbf{C} , let E be an effect theory and let an interpretation of the value theory of E in \mathbf{V} be given. The following data are equivalent:*

- (1) A comodel structure for E on \underline{S}
- (2) An interpretation of E in the Kleisli model $(\mathbf{V}, \mathbf{Kl}_{\underline{S}}, J_{\underline{S}})$

- (3) For each effect constant $e : \bar{\beta}; \bar{\alpha}_1 + \dots + \bar{\alpha}_n$ in E an algebraic operation: a \mathbf{V} -natural family of morphisms in \mathbf{V}

$$\left\{ \prod_{i=1}^n (U_{\underline{S}} \underline{X})^{[\bar{\alpha}_i]} \rightarrow (U_{\underline{S}} \underline{X})^{[\bar{\beta}]} \right\}_{\underline{X} \in \mathbf{C}}$$

equipping each $U_{\underline{S}}(\underline{X}) \stackrel{\text{def}}{=} \mathbf{C}(\underline{S}, \underline{X})$ with a model structure for E .

Proof. We first prove equivalence of (1) and (2). First note that in both cases, an effect constant $e : \bar{\beta}; \bar{\alpha}_1 + \dots + \bar{\alpha}_n$ is modelled as a morphism

$$[[e]] : [[\bar{\beta}]] \cdot \underline{S} \rightarrow ([[\bar{\alpha}_1] + \dots + [\bar{\alpha}_n]]) \cdot \underline{S}.$$

It thus suffices to show that for any term $\Gamma \vdash^p M : \sigma$ of the fragment of Figure 6 the morphisms $[[M_s^{\underline{S}}]], [[M]] : [[\Gamma]] \cdot \underline{S} \rightarrow [[\sigma]] \cdot \underline{S}$ are equal, where $[[M_s^{\underline{S}}]]$ is the ECBV term $M_s^{\underline{S}}$ interpreted in the enriched model $(\mathbf{V}, \mathbf{C}, \underline{S})$ and $[[M]]$ is the fine-grain call-by-value term M interpreted in the Kleisli model $(\mathbf{V}, \mathbf{Kl}_{\underline{S}}, J_{\underline{S}})$. This can be proved by induction on the structure of M .

To prove equivalence of (1) and (3) we build on the equivalence of state access operations and algebraic operations of Theorem 8.1. First we show that for any term $\Gamma \vdash^p M : \sigma$ of the fragment of Figure 6 the equation

$$\mathbf{C}([[M_s^{\underline{S}}]], \underline{X}) = [[M]] : \mathbf{C}([[\sigma]], \underline{S}, \underline{X}) \rightarrow \mathbf{C}([[\Gamma]], \underline{S}, \underline{X}) \quad (8.14)$$

holds. This time the denotation brackets on the left hand side refers to the interpretation of ECBV in $(\mathbf{V}, \mathbf{C}, \underline{S})$, and the denotation brackets on the right hand side refer to the interpretation of fine-grain call-by-value in the Kleisli model $(\mathbf{V}, \mathbf{Kl}_{R R(-)}, J)$, where $R = U_{\underline{S}}(\underline{X})$. As above, $[[M_s^{\underline{S}}]]$ is simply the interpretation of M in the Kleisli model $(\mathbf{V}, \mathbf{Kl}_{\underline{S}}, J_{\underline{S}})$. Equation (8.14) can be proved by induction on the structure of M . (There is also a categorical perspective on this, based around the construction $\mathbf{C}(-, \underline{X})$ which gives rise to an identity-on-objects functor $\mathbf{Kl}_{\underline{S}} \rightarrow \mathbf{Kl}_{R R(-)}$ that preserves sums and the action of \mathbf{V} , although it does not preserve the enrichment.)

From (8.14) we deduce the equivalence of (1) and (3). In fact (1) \implies (3) is immediate: suppose \underline{S} is a comodel, and consider an equation $\Gamma \vdash M \equiv N : \tau$ in the theory. Since \underline{S} is a comodel, we have $[[M_s^{\underline{S}}]] = [[N_s^{\underline{S}}]]$ and so $[[M]] = [[N]]$ as interpreted in $(\mathbf{V}, \mathbf{Kl}_{R R(-)}, J)$.

For (3) \implies (1), suppose that $R = \mathbf{C}(\underline{S}, \underline{X})$ is a model for every \underline{X} , naturally in \underline{X} . Then by (8.14)

$$\mathbf{C}([[M_s^{\underline{S}}]], \underline{X}) = \mathbf{C}([[N_s^{\underline{S}}]], \underline{X}) : \mathbf{C}([[\tau]], \underline{S}, \underline{X}) \rightarrow \mathbf{C}([[\Gamma]], \underline{S}, \underline{X})$$

holds for all equations $\Gamma \vdash M \equiv N : \tau$ and all \underline{X} . The enriched Yoneda embedding is full and faithful and so $[[M_s^{\underline{S}}]] = [[N_s^{\underline{S}}]] : [[\Gamma]] \cdot \underline{S} \rightarrow [[\tau]] \cdot \underline{S}$, proving that \underline{S} is a comodel.

We remark that the equivalence of (2) and (3) is in the spirit of [38, §6]. \square

8.8. Generalizing full completeness to the case of effects. The full completeness result of Theorem 4.3 extends verbatim to the case of the calculi augmented with an effect theory E . The proof is based on an extension of the coreflection theorem (Theorem 5.3) which we state below.

First 2-categories $\mathfrak{d}\mathfrak{Kl}eisl_E$ and $\mathfrak{d}\mathfrak{E}cbv_E$ of distributive Kleisli models of E and distributive enriched models of E are defined. These are defined similarly to $\mathfrak{Kl}eisl$ and $\mathfrak{E}cbv$ except

morphisms are required to preserve coproducts (up to isomorphism) and the interpretation of E (strictly). Details can be found in Appendix A.5.

Lemma 8.3. *The assignments $\mathbf{St}(\mathbf{V}, \mathbf{C}, J) \stackrel{\text{def}}{=} (\mathbf{V}, \mathbf{C}, 1)$ and $\mathbf{Kl}(\mathbf{V}, \mathbf{C}, \underline{S}) \stackrel{\text{def}}{=} (\mathbf{V}, \mathbf{Kl}_S, J_S)$ extend to 2-functors*

$$\mathbf{St}: \mathfrak{d}\mathfrak{Kl}eisli_E \rightarrow \mathfrak{d}\mathfrak{E}cbv_E$$

$$\mathbf{Kl}: \mathfrak{d}\mathfrak{E}cbv_E \rightarrow \mathfrak{d}\mathfrak{Kl}eisli_E$$

Proof (sketch). We just show that these are well-defined on objects. The case of \mathbf{Kl} is simply the implication from (1) to (2) of Theorem 8.2.

In the case of \mathbf{St} we must show that 1 carries a comodel structure for E whenever $(\mathbf{V}, \mathbf{C}, J)$ models E . An effect constant $e : \bar{\beta}; \bar{\alpha}_1 + \dots + \bar{\alpha}_n$ can be modelled as the composite

$$\llbracket \bar{\beta} \rrbracket \cdot 1 \xrightarrow{J(\pi_1)} \llbracket \bar{\beta} \rrbracket \xrightarrow{\llbracket e \rrbracket} \llbracket \bar{\alpha}_1 + \dots + \bar{\alpha}_n \rrbracket \xrightarrow{J(\langle id, ! \rangle)} \llbracket \bar{\alpha}_1 + \dots + \bar{\alpha}_n \rrbracket \cdot 1$$

where $\llbracket e \rrbracket$ refers to the interpretation of e in the given E -model structure of $(\mathbf{V}, \mathbf{C}, J)$. We must show that this defines a comodel, i.e., that the equations are satisfied. To this end one can prove that for any term $\Gamma \vdash^p M : \sigma$ of the fragment of fine-grain call-by-value used for effect theories (Figure 6) the equation $\llbracket M_s^S \rrbracket = J(\langle id, ! \rangle) \circ \llbracket M \rrbracket \circ J(\pi_1)$ holds. Here, on the left hand side the double brackets refer to the interpretation of ECBV in $(\mathbf{V}, \mathbf{C}, 1)$ and the brackets on the right hand side refer to the interpretation of fine-grain call-by-value in $(\mathbf{V}, \mathbf{C}, J)$. This equation is proved by induction on typing derivations. Thus, for any equation $\Gamma \vdash^p M \equiv N : \tau$ in E , since $(\mathbf{V}, \mathbf{C}, J)$ models E we have $\llbracket M \rrbracket = \llbracket N \rrbracket$, and thus also $\llbracket M_s^S \rrbracket = \llbracket N_s^S \rrbracket$, proving that 1 is indeed a comodel. \square

We end this section by stating the coreflection theorem for models of effect theories.

Theorem 8.4. *The 2-functor $\mathbf{St}: \mathfrak{d}\mathfrak{Kl}eisli_E \rightarrow \mathfrak{d}\mathfrak{E}cbv_E$ is left biadjoint to \mathbf{Kl} , i.e., for any pair of objects $(\mathbf{V}, \mathbf{C}, J)$ and $(\mathbf{V}', \mathbf{C}', \underline{S})$ of $\mathfrak{d}\mathfrak{Kl}eisli_E$ and $\mathfrak{d}\mathfrak{E}cbv_E$ respectively, there is an equivalence of categories*

$$\mathfrak{d}\mathfrak{E}cbv_E(\mathbf{St}(\mathbf{V}, \mathbf{C}, J), (\mathbf{V}', \mathbf{C}', \underline{S})) \simeq \mathfrak{d}\mathfrak{Kl}eisli_E((\mathbf{V}, \mathbf{C}, J), \mathbf{Kl}(\mathbf{V}', \mathbf{C}', \underline{S}))$$

natural in $(\mathbf{V}, \mathbf{C}, J)$ and $(\mathbf{V}', \mathbf{C}', \underline{S})$. Moreover, the unit of the adjunction $\eta: id_{\mathfrak{d}\mathfrak{E}cbv_E} \rightarrow \mathbf{Kl} \circ \mathbf{St}$ is an isomorphism.

9. RELATIONSHIP WITH ATKEY'S PARAMETERIZED MONADS

Atkey's work on parameterized monads [3], has proven relevant to functional programming (e.g. [24, §5.2]). In this section we show that parameterized monads are essentially the same as enriched models.

Recall that, in general category theory, if a functor $F : \mathcal{A} \times \mathcal{S} \rightarrow \mathcal{B}$ is such that $F(-, S) : \mathcal{A} \rightarrow \mathcal{B}$ has a right adjoint $G(S, -) : \mathcal{B} \rightarrow \mathcal{A}$ for each S , then these right adjoints together form a functor $G : \mathcal{S}^{\text{op}} \times \mathcal{B} \rightarrow \mathcal{A}$ called the parameterized right adjoint. Atkey has carried out a study of a generalized form of monad that arises from parameterized adjunctions: the functor $G(-_1, F(-_2, -_3)) : \mathcal{S}^{\text{op}} \times \mathcal{A} \times \mathcal{S} \rightarrow \mathcal{A}$ is called a parameterized

monad. Thus a parameterized monad is a functor $T : \mathcal{S}^{\text{op}} \times \mathcal{A} \times \mathcal{S} \rightarrow \mathcal{A}$ together with extranatural families of morphisms

$$\begin{aligned} \eta_{S,A} &: A \rightarrow T(S, A, S) \\ \mu_{S_1, S_2, S_3, A} &: T(S_1, T(S_2, A, S_3), S_2) \rightarrow T(S_1, A, S_3) \end{aligned}$$

satisfying monad laws. A first example of a parameterized monad is the parameterized state monad on the category of sets: $T(S_1, A, S_2) \stackrel{\text{def}}{=} [S_1 \Rightarrow A \times S_2]$.

Every enriched model (\mathbf{V}, \mathbf{C}) contains a parameterized adjunction, since $\mathbf{C}(-_1, -_2) : \mathbf{C}^{\text{op}} \times \mathbf{C} \rightarrow \mathbf{V}$ is by definition a parameterized right adjoint for $(-)_1 \cdot (-)_2 : \mathbf{V} \times \mathbf{C} \rightarrow \mathbf{C}$.

Conversely, in the theory of parameterized monads, the following Kleisli construction [3, Prop. 1] plays a key role. Given a parameterized monad $T : \mathcal{S}^{\text{op}} \times \mathcal{A} \times \mathcal{S} \rightarrow \mathcal{A}$, the objects of the Kleisli category are pairs (A, S) of an object of \mathcal{A} and an object of \mathcal{S} , and a morphism $(A, S) \rightarrow (A', S')$ is a morphism $A \rightarrow T(S, A', S')$ in \mathbf{V} . This is a first step towards building an enriched model from a parameterized monad.

Plain parameterized monads are not especially relevant to the theory of programming languages, just as plain monads are not very relevant. In his study of parameterized monads, Atkey focuses on *strong parameterized monads with Kleisli exponentials* [3, §2.4.1]. He uses these to provide semantics for a ‘command calculus’, which is a term language closely related to our basic enriched call-by-value calculus (Figure 1).

Proposition 9.1. *Let \mathbf{V} be a category with finite products. The following data are equivalent.*

- (1) *A strong parameterized monad on \mathbf{V} with Kleisli exponentials, taking parameters in a category \mathcal{S} [3, §2.4.1].*
- (2) *A category \mathbf{C} enriched in \mathbf{V} with copowers (i.e., an enriched model – §2.2) with a chosen subcategory \mathcal{S} of \mathbf{C} such that every object \underline{X} of \mathbf{C} is of the form $\underline{X} = A \cdot \underline{S}$ for A in \mathbf{V} and \underline{S} in \mathcal{S} .*

Proof notes. Given a strong parameterized monad $T : \mathcal{S}^{\text{op}} \times \mathbf{V} \times \mathcal{S} \rightarrow \mathbf{V}$, we let \mathbf{C} be the Kleisli category for T , as above. The pair (\mathbf{V}, \mathbf{C}) forms an enriched model with $A \cdot (B, \underline{S}) \stackrel{\text{def}}{=} (A \times B, \underline{S})$: this is a rephrasing of what it means for T to be strong and have Kleisli exponentials. Moreover \mathcal{S} can be identified with a subcategory of \mathbf{C} whose objects are of the form $(1, \underline{S})$ and whose morphisms are induced by the morphisms in \mathcal{S} .

Conversely, suppose we are given an enriched model (\mathbf{V}, \mathbf{C}) and a chosen subcategory \mathcal{S} of \mathbf{C} . We define a parameterized monad $T : \mathcal{S}^{\text{op}} \times \mathbf{V} \times \mathcal{S} \rightarrow \mathbf{V}$ by

$$T(\underline{S}, A, \underline{S}') \stackrel{\text{def}}{=} \mathbf{C}(\underline{S}, A \cdot \underline{S}') \quad .$$

It is routine to check that the two constructions are mutually inverse, up-to equivalence of categories. \square

10. RELATIONSHIP WITH THE ENRICHED EFFECT CALCULUS

Our enriched call-by-value calculus (ECBV) is a fragment of the enriched effect calculus (EEC, [8, 7]) which was designed to analyze linear usage in effectful computation. We now show that ECBV is not only a syntactic fragment of EEC: every model of the enriched call-by-value calculus embeds in a model of the enriched effect calculus.

The enriched effect calculus extends the enriched call-by-value calculus that we introduced in Section 2 with some type constructions:

$$\begin{array}{l}
 \mathbf{A}, \mathbf{B} ::= \alpha \mid 1 \mid \mathbf{A} \times \mathbf{B} \mid \underline{\mathbf{A}} \multimap \underline{\mathbf{B}} \quad \mid \quad 0 \mid \mathbf{A} + \mathbf{B} \quad \mid \quad \mathbf{A} \rightarrow \mathbf{B} \mid \underline{\alpha} \mid \underline{0} \mid \underline{\mathbf{A}} \oplus \underline{\mathbf{B}} \mid !\mathbf{A} \otimes \underline{\mathbf{B}} \mid !\mathbf{A} \\
 \underline{\mathbf{A}}, \underline{\mathbf{B}} ::= \underline{\alpha} \mid !\mathbf{A} \otimes \underline{\mathbf{B}} \quad \mid \quad \underline{0} \mid \underline{\mathbf{A}} \oplus \underline{\mathbf{B}} \quad \mid \quad \underline{1} \mid \underline{\mathbf{A}} \times \underline{\mathbf{B}} \mid \underline{\mathbf{A}} \rightarrow \underline{\mathbf{B}} \mid !\mathbf{A} .
 \end{array}$$

Figure 1
Figure 3
Full EEC [8]

The additional types are: products ($\underline{\mathbf{A}} \times \underline{\mathbf{B}}$) and powers ($\mathbf{A} \rightarrow \underline{\mathbf{B}}$) of computation types; an operation to coerce a value type \mathbf{A} into a computation type $!\mathbf{A}$; a space of pure functions ($\mathbf{A} \rightarrow \mathbf{B}$) between value types; and an implicit inclusion of computation types as value types.

These additional types have been used to describe other aspects of effectful computation. We briefly considered a linear-use CPS translation in Section 7, based on [9, 10], for which we needed products and powers of computation types. Egger et al. [8] also describe monadic call-by-name and call-by-value interpretations, for which they use the coercion of value types into computation types and the implicit inclusion of computation types in value types.

The additional types of EEC do not affect the full completeness of the linear state-passing translation (Thm. 4.3), for the following reason. In Theorem 10.2 we show that every model of ECBV embeds in a model of EEC; conservativity of EEC over ECBV then follows from a strong normalisation result for EEC [7]. Thus the linear-use state-passing translation of Section 4 can be understood as a fully complete translation into EEC.

Definition 10.1. A *closed enriched model* is a pair of categories (\mathbf{V}, \mathbf{C}) such that \mathbf{V} is cartesian closed with coproducts and \mathbf{C} is \mathbf{V} -enriched with powers and copowers and products and coproducts.

A *model of EEC* $(\mathbf{V}, \mathbf{C}, F, U)$ [8] is a closed enriched model (\mathbf{V}, \mathbf{C}) together with a \mathbf{V} -enriched adjunction $F \dashv U : \mathbf{C} \rightarrow \mathbf{V}$.

We refer to [8] for the term calculus and interpretation of EEC in EEC models. Here, we will analyze how the notion of EEC model compares to the other notions of model that we have discussed so far. One simple observation is that every closed enriched model is a distributive enriched model in the sense of Definition 6.1. Another observation is that the adjunction part of an EEC model can be equivalently given by a ‘state’ object of \mathbf{C} (see [8, Proof of Thm. 4] and Section 5.2).

Theorem 10.2. *Every enriched model embeds in a closed enriched model.*

Proof. The difference between enriched models and closed enriched models is that in an enriched model (\mathbf{V}, \mathbf{C}) the value category \mathbf{V} need not be cartesian closed nor have coproducts, and the computation category \mathbf{C} need not have coproducts, products and powers.

We use the Yoneda embedding to embed an enriched model in a closed enriched model. For any small category \mathcal{A} we consider the category $\hat{\mathcal{A}}$ of contravariant functors, $\mathcal{A}^{\text{op}} \rightarrow \mathbf{Set}$, and natural transformations between them. The Yoneda embedding $\mathcal{A} \mapsto \mathcal{A}(-, A)$ is a functor $\mathbf{y}_{\mathcal{A}} : \mathcal{A} \rightarrow \hat{\mathcal{A}}$ that exhibits $\hat{\mathcal{A}}$ as a cocompletion of \mathcal{A} . That is: $\hat{\mathcal{A}}$ is cocomplete, with colimits computed pointwise, and for any other cocomplete category \mathcal{B} and any functor $F : \mathcal{A} \rightarrow \mathcal{B}$ there is a colimit-preserving functor $F_! : \hat{\mathcal{A}} \rightarrow \mathcal{B}$ given by $F_!(P) \stackrel{\text{def}}{=} \text{colim}((\mathbf{y}_{\mathcal{A}} \downarrow P) \xrightarrow{\pi} \mathcal{A} \xrightarrow{F} \mathcal{B})$, where $(\mathbf{y}_{\mathcal{A}} \downarrow P)$ is the category of elements of P ; this colimit-preserving functor is essentially unique such that $F \cong F_! \cdot \mathbf{y}_{\mathcal{A}}$.

Let (\mathbf{V}, \mathbf{C}) be an enriched model. We will show that $(\hat{\mathbf{V}}, \hat{\mathbf{C}})$ is a closed enriched model, and that (\mathbf{V}, \mathbf{C}) embeds in it as an enriched model.

We proceed by considering the following 2-categorical situation. Because the construction $\widehat{(-)}$ is a free cocompletion, it can be understood as a weak 2-functor from the 2-category \mathbf{Cat} of small categories, functors and natural transformations to the 2-category \mathbf{Cocomp} of categories with all colimits, colimit-preserving functors and natural transformations.

In fact it is necessary to be slightly more general than this: we will understand \mathbf{Cat} and \mathbf{Cocomp} as 2-multicategories. Recall that a 2-multicategory is a \mathbf{Cat} -enriched multicategory. So it is like a 2-category except that the domains of the 1-cells are sequences of objects.

- The 2-multicategory \mathbf{Cat} is defined as follows. The 0-cells are small categories with finite coproducts. The 1-cells $F: (\mathcal{A}_1, \dots, \mathcal{A}_n) \rightarrow \mathcal{B}$ in \mathbf{Cat} are functors in n arguments, i.e. functors $F: \mathcal{A}_1 \times \dots \times \mathcal{A}_n \rightarrow \mathcal{B}$. The 2-cells are natural transformations.
- The objects of the 2-multicategory \mathbf{Cocomp} are categories with all colimits. The 1-cells $F: (\mathcal{A}_1, \dots, \mathcal{A}_n) \rightarrow \mathcal{B}$ in \mathbf{Cocomp} are functors $F: \mathcal{A}_1 \times \dots \times \mathcal{A}_n \rightarrow \mathcal{B}$ that preserve colimits in each argument, i.e. that for fixed $A_1 \in \mathcal{A}_1, \dots, A_n \in \mathcal{A}_n$ and for $1 \leq i \leq n$, the functor $F(A_1, \dots, -_i, \dots, A_n): \mathcal{A}_i \rightarrow \mathcal{B}$ preserves colimits. The 2-cells are natural transformations.
- The construction $\widehat{(-)}$ extends to a weak morphism of 2-multicategories from \mathbf{Cat} to \mathbf{Cocomp} . A 1-cell $F: (\mathcal{A}_1, \dots, \mathcal{A}_n) \rightarrow \mathcal{B}$ in \mathbf{Cat} is extended to a 1-cell in \mathbf{Cocomp} , i.e. a functor $F_! : \widehat{\mathcal{A}_1} \times \dots \times \widehat{\mathcal{A}_n} \rightarrow \widehat{\mathcal{B}}$ which preserves colimits in each argument. This construction is done by iteratively applying the following idea. If an n -ary functor $G: (\mathcal{A}_1, \dots, \mathcal{A}_k, \dots, \mathcal{A}_n) \rightarrow \mathcal{B}$ is such that \mathcal{B} is cocomplete, $\mathcal{A}_1 \dots \mathcal{A}_{k-1}$ are cocomplete, G preserves colimits in each of the first $(k-1)$ arguments, and \mathcal{A}_k is small, then there is an n -ary functor $G_{!k}: (\mathcal{A}_1, \dots, \widehat{\mathcal{A}_k}, \dots, \mathcal{A}_n) \rightarrow \mathcal{B}$ that preserves colimits in each of the first k arguments such that $G \cong G_{!k} \cdot (\mathcal{A}_1, \dots, \mathbf{y}_{\mathcal{A}_k}, \dots, \mathcal{A}_n)$. This is because the n -ary functor $G: (\mathcal{A}_1, \dots, \mathcal{A}_n) \rightarrow \mathcal{B}$ can be carried to a functor

$$\mathcal{A}_k \rightarrow \mathbf{Cocomp}(\mathcal{A}_1, \dots, \mathcal{A}_{k-1}; \mathbf{Cat}(\mathcal{A}_{k+1}, \dots, \mathcal{A}_n; \mathcal{B}))$$

whose codomain is cocomplete, and which can thus be extended to a colimit-preserving functor using the universal property of $\widehat{\mathcal{A}_k}$:

$$\widehat{\mathcal{A}_k} \rightarrow \mathbf{Cocomp}(\mathcal{A}_1, \dots, \mathcal{A}_{k-1}; \mathbf{Cat}(\mathcal{A}_{k+1}, \dots, \mathcal{A}_n; \mathcal{B}));$$

this can be uncurried to give $G_{!k}: (\mathcal{A}_1, \dots, \widehat{\mathcal{A}_k}, \dots, \mathcal{A}_n) \rightarrow \mathcal{B}$. Ultimately, the extension $F_! : \widehat{\mathcal{A}_1} \times \dots \times \widehat{\mathcal{A}_n} \rightarrow \widehat{\mathcal{B}}$ satisfies the following coend formula:

$$F_!(P_1, \dots, P_n)(B) \cong \int^{\mathcal{A}_1, \dots, \mathcal{A}_n} P_1(\mathcal{A}_1) \times \dots \times P_n(\mathcal{A}_n) \times \mathcal{B}(B, F(\mathcal{A}_1, \dots, \mathcal{A}_n))$$

- Recall the following consequence of the special adjoint functor theorem: a morphism $F: (\widehat{\mathcal{A}_1}, \dots, \widehat{\mathcal{A}_n}) \rightarrow \mathcal{B}$ in \mathbf{Cocomp} can be equivalently described as a functor that has a right adjoint in each argument, i.e. a right adjoint for each functor $F(P_1, \dots, -_i, \dots, P_n): \widehat{\mathcal{A}_i} \rightarrow \mathcal{B}$.
- Aside from size issues, there is a forgetful morphism of 2-multicategories $\mathbf{Cat} \rightarrow \mathbf{Cocomp}$ and $\widehat{(-)}$ is left biadjoint to it. The Yoneda embedding is the unit for this adjunction.

With the general situation explained, the proof of Theorem 10.2 is straightforward. We begin by considering the evident notion of ‘weak monoid’ in a 2-multicategory \mathfrak{K} . This comprises an object \mathcal{M} of \mathfrak{K} and two 1-cells: $m: (\mathcal{M}, \mathcal{M}) \rightarrow \mathcal{M}$ and $e: () \rightarrow \mathcal{M}$, with three coherence 2-isomorphisms. A morphism of 2-multicategories $\mathfrak{K} \rightarrow \mathfrak{K}'$ takes weak monoids in \mathfrak{K} to weak monoids in \mathfrak{K}' . In particular a monoidal category is a weak monoid in \mathbf{Cat} , and the construction $\widehat{(-)}$ takes it to a weak monoid in \mathbf{Cocomp} , which is a cocomplete biclosed

monoidal category. The Yoneda embedding $\mathcal{M} \rightarrow \widehat{\mathcal{M}}$ preserves the weak monoid structure. This is Day’s convolution construction [5, 17].

In particular, the value category \mathbf{V} of our enriched model has products and this exhibits it as a monoidal category. It follows that $\widehat{\mathbf{V}}$ is cartesian closed and that the Yoneda embedding $\mathbf{V} \rightarrow \widehat{\mathbf{V}}$ preserves the product structure in \mathbf{V} . Given a weak monoid \mathcal{M} in a 2-multicategory \mathfrak{K} , we consider the evident notion of weak action for \mathcal{M} : an object \mathcal{A} of \mathfrak{K} and a 1-cell $(\mathcal{M}, \mathcal{A}) \rightarrow \mathcal{A}$ satisfying the laws of monoid actions up-to coherent isomorphism. A morphism of 2-multicategories takes weak monoid actions to weak monoid actions. In particular, given a monoidal category \mathcal{M} , an action of \mathcal{M} on another category \mathcal{A} induces an enrichment of $\widehat{\mathcal{A}}$ in $\widehat{\mathcal{M}}$ with powers and copowers. The Yoneda embedding $\mathcal{A} \rightarrow \widehat{\mathcal{A}}$ preserves the monoidal action. Moreover since it is 2-natural it preserves any enrichment or powers that already exist in \mathcal{A} .

In particular, in our enriched model, \mathbf{V} acts on \mathbf{C} and so $\widehat{\mathbf{C}}$ is enriched in $\widehat{\mathbf{V}}$ with powers and copowers, and the Yoneda embedding $\mathbf{C} \rightarrow \widehat{\mathbf{C}}$ is enriched in \mathbf{V} and preserves copowers. \square

The crux of the proof is that the Yoneda embedding adds closed structure — cartesian closed structure and powers — while preserving the other structure. Although the Yoneda embedding does not *freely* add the closed structure, it is considerably simpler than the free closure. This is one reason why Yoneda embeddings are a common technique in semantics. For instance, the enriched Yoneda embedding is used by Egger et al. [8] to show that Levy’s call-by-push-value embeds in the enriched effect calculus (but the *enriched* Yoneda embedding is not appropriate in our proof because it does not preserve copowers).

Theorem 10.2 explains that EEC is conservative over ECBV, but it neglects sum types. Sum types play an important role in the study of generic effects and state access operations (§8). We now show that EEC is conservative over ECBV with sum types.

Proposition 10.3. *Every distributive enriched model (§6.1) embeds in a closed enriched model (Def. 10.1).*

Proof. Our proof of Proposition 10.3 follows the same outline as our proof of Theorem 10.2. We must modify that proof because the Yoneda embedding $\mathbf{y}_{\mathcal{A}} : \mathcal{A} \rightarrow \widehat{\mathcal{A}}$ does not preserve coproducts. We use the following variation on the Yoneda embedding. For any category \mathcal{A} with finite coproducts, let $\text{FP}(\mathcal{A}^{\text{op}}, \mathbf{Set})$ be the category of finite-product-preserving functors $\mathcal{A}^{\text{op}} \rightarrow \mathbf{Set}$ and natural transformations between them. Assuming \mathcal{A} has coproducts, the category $\text{FP}(\mathcal{A}^{\text{op}}, \mathbf{Set})$ has coproducts too. The Yoneda embedding $A \mapsto \mathcal{A}(-, A)$ is a functor $\mathcal{A} \rightarrow \text{FP}(\mathcal{A}^{\text{op}}, \mathbf{Set})$ which preserves coproducts. In fact, the Yoneda embedding exhibits $\text{FP}(\mathcal{A}^{\text{op}}, \mathbf{Set})$ as the cocompletion of \mathcal{A} as a category with finite coproducts. The category $\text{FP}(\mathcal{A}^{\text{op}}, \mathbf{Set})$ is cocomplete (although not all colimits are computed pointwise), and for any coproduct-preserving functor $F : \mathcal{A} \rightarrow \mathcal{B}$ into a cocomplete category \mathcal{B} there is a colimit-preserving functor $F_! : \text{FP}(\mathcal{A}^{\text{op}}, \mathbf{Set}) \rightarrow \mathcal{B}$ given by $F_!(P) \stackrel{\text{def}}{=} \text{colim}((\mathbf{y}_{\mathcal{A}} \downarrow P) \xrightarrow{\pi} \mathcal{A} \xrightarrow{F} \mathcal{B})$; this colimit-preserving functor is essentially unique such that $F \cong F_! \cdot \mathbf{y}_{\mathcal{A}}$ (see e.g. [21, Thms 5.86, 6.11], [42], [11]). Since a distributive enriched model has coproducts, this is the right variation of the Yoneda embedding to use.

We now mimic the proof of Theorem 10.2, replacing the cocompletion construction $\widehat{(-)}$ with the cocompletion $\text{FP}((-)^{\text{op}}, \mathbf{Set})$ of a category with coproducts. Consider the 2-multicategory $\mathbf{Coproduct}$: the 0-cells are small categories with finite coproducts; the 1-cells $F : (\mathcal{A}_1, \dots, \mathcal{A}_n) \rightarrow \mathcal{B}$ are functors $F : \mathcal{A}_1 \times \dots \times \mathcal{A}_n \rightarrow \mathcal{B}$ that preserve coproducts in each

argument; the 2-cells are natural transformations. The construction $\text{FP}((-)^{\text{op}}, \mathbf{Set})$ extends to a morphism of 2-multicategories from $\mathbf{Coproduct}$ to \mathbf{Cocomp} . By the special adjoint functor theorem, a morphism $(\text{FP}(\mathcal{A}_1^{\text{op}}, \mathbf{Set}), \dots, \text{FP}(\mathcal{A}_n^{\text{op}}, \mathbf{Set})) \rightarrow \mathcal{B}$ in \mathbf{Cocomp} is a functor that has a right adjoint in each argument.

A weak monoid \mathcal{M} in $\mathbf{Coproduct}$ is a distributive monoidal category, i.e., a monoidal category with coproducts such that the tensor preserves coproducts in each argument. The construction $\text{FP}(-^{\text{op}}, \mathbf{Set})$ takes it to a weak monoid in \mathbf{Cocomp} , which is a cocomplete biclosed monoidal category. The Yoneda embedding $\mathcal{M} \rightarrow \text{FP}(\mathcal{M}^{\text{op}}, \mathbf{Set})$ preserves the weak monoid structure and coproducts.

In particular, if (\mathbf{V}, \mathbf{C}) is a distributive enriched model then \mathbf{V} has distributive products and this exhibits it as a distributive monoidal category. It follows that $\text{FP}(\mathbf{V}^{\text{op}}, \mathbf{Set})$ is cartesian closed with coproducts and that the Yoneda embedding $\mathbf{V} \rightarrow \text{FP}(\mathbf{V}^{\text{op}}, \mathbf{Set})$ preserves the coproduct and product structure in \mathbf{V} .

An action of a weak monoid in $\mathbf{Coproduct}$ is the same thing as a distributive action in the sense of Section 6.1. Given a distributive monoidal category \mathcal{M} , a distributive action of \mathcal{M} on a category \mathcal{A} with finite coproducts induces an enrichment of $\text{FP}(\mathcal{A}^{\text{op}}, \mathbf{Set})$ in $\text{FP}(\mathcal{M}^{\text{op}}, \mathbf{Set})$ with powers and copowers. The Yoneda embedding $\mathcal{A} \rightarrow \text{FP}(\mathcal{A}^{\text{op}}, \mathbf{Set})$ preserves coproducts and the monoidal action. Moreover since it is 2-natural it preserves any enrichment or powers that already exist in \mathcal{A} .

In particular, if (\mathbf{V}, \mathbf{C}) is a distributive enriched model then \mathbf{V} acts on \mathbf{C} , and so $\text{FP}(\mathbf{C}^{\text{op}}, \mathbf{Set})$ is enriched in $\text{FP}(\mathbf{V}^{\text{op}}, \mathbf{Set})$ with powers and copowers, and the Yoneda embedding $\mathbf{C} \rightarrow \text{FP}(\mathbf{C}^{\text{op}}, \mathbf{Set})$ is enriched in \mathbf{V} and preserves coproducts and copowers. \square

The construction in this proof is related to the following natural situation. Let \mathbf{Set}_f be the category of finite sets, and let \mathbb{T} be a Lawvere theory. Then $(\mathbf{Set}_f, \mathbb{T}^{\text{op}})$ is almost an enriched model, except that the category \mathbb{T}^{op} is typically not \mathbf{Set}_f -enriched. Nonetheless, our construction applied to $(\mathbf{Set}_f, \mathbb{T}^{\text{op}})$ yields the basic motivating example of an EEC model: $\text{FP}(\mathbf{Set}_f^{\text{op}}, \mathbf{Set})$ is the category of sets (since \mathbf{Set}_f is the free category with finite coproducts on one generator) and $\text{FP}(\mathbb{T}, \mathbf{Set})$ is the category of algebras of the Lawvere theory. (See also [41, Thm. 38].)

REFERENCES

- [1] Peter Achten and Marinus J. Plasmeijer. The ins and outs of Clean I/O. *J. Funct. Program.*, 5(1):81–110, 1995.
- [2] J Aczél. On mean values. *Bull. Amer. Math. Soc.*, 54(4):392–400, 1948.
- [3] Robert Atkey. Parameterised notions of computation. *J. Funct. Program.*, 19(3–4):335–376, 2009.
- [4] Josh Berdine, Peter W. O’Hearn, Uday S. Reddy, and Hayo Thielecke. Linear continuation-passing. *Higher-Order and Symbolic Computation*, 15(2-3):181–208, 2002.
- [5] Brian Day. On closed categories of functors. In *Lect. Notes Math. 137*, pages 1–38. Springer, 1970.
- [6] Eduardo J Dubuc and G.M Kelly. A presentation of topoi as algebraic relative to categories or graphs. *J. Algebra*, 81(2):420 – 433, 1983.
- [7] J. Egger, R.E. Møgelberg, and A. Simpson. The enriched effect calculus: syntax and semantics. To appear in *Journal of Logic and Computation*.
- [8] J. Egger, R.E. Møgelberg, and A. Simpson. Enriching an effect calculus with linear types. In *CSL’09*, pages 240–254. Springer, 2009.
- [9] J. Egger, R.E. Møgelberg, and A. Simpson. Linearly-used continuations in the enriched effect calculus. In *Proc. FOSSACS’10*, volume 6014, pages 18–32. Springer, 2010.

- [10] J. Egger, R.E. Møgelberg, and A. Simpson. Linear-use CPS translations in the enriched effect calculus. *Logical Methods in Computer Science*, 8(4), 2012.
- [11] Marcelo P. Fiore. Enrichment and representation theorems for categories of domains and continuous functions. Unpublished manuscript, March 1996.
- [12] Carsten Führmann. Varieties of effects. In *Proc. FOSSACS'02*, pages 144–158, 2002.
- [13] Jean-Yves Girard. Linear logic. *Theor. Comput. Sci.*, 50:1–102, 1987.
- [14] R. Gordon and A.J. Power. Enrichment through variation. *J. Pure Appl. Algebra*, 120:167–185, 1997.
- [15] M. Hasegawa. Linearly used effects: Monadic and CPS transformations into the linear lambda calculus. In *Proc. 6th International Symposium on Functional and Logic Programming (FLOPS)*, volume 2441 of *LNCS*, pages 167–182. Springer, 2002.
- [16] Reinhold Heckmann. Probabilistic domains. In *Proc. Trees in Algebra and Programming – CAAP'94*, volume 787 of *Lecture Notes in Computer Science*, pages 142–156. Springer, 1994.
- [17] Geun Bin Im and G M Kelly. A universal property of the convolution monoidal structure. *J. Pure Appl. Alg.*, 43:75–88, 1986.
- [18] G. Janelidze and G. M. Kelly. A note on actions of a monoidal category. *Theory Appl. of Categ.*, 9(4):61–91, 2001.
- [19] Alan Jeffrey. Premonoidal categories and a graphical view of programs. Available at <ftp://outside.cs.bell-labs.com/who/ajeffrey/papers/premonA.pdf>, 1997.
- [20] G. M. Kelly. Adjunction for enriched categories. In *Lect. Notes Math. 106*, pages 166–177. Springer, 1969.
- [21] G. M. Kelly. *Basic Concepts of Enriched Category Theory*. Cambridge University Press, 1982.
- [22] G M Kelly. Elementary observations on 2-categorical limits. *Bull. Austral. Math. Soc.*, 39(2):301317, 1989.
- [23] G M Kelly and A J Power. Adjunctions whose counits are coequalizers, and presentations of finitary enriched monads. *J. Pure Appl. Algebra*, 89:163–179, 1993.
- [24] Oleg Kiselyov, Simon Peyton Jones, and C. Shan. Fun with type functions. In *Reflections on the Work of C.A.R. Hoare*, pages 301–331. Springer, 2010.
- [25] Anders Kock. Bilinearity and cartesian closed monads. *Math. Scand.*, 29:161–174, 1971.
- [26] P. B. Levy. *Call By Push Value*. Kluwer, December 2003.
- [27] P.B. Levy, J. Power, and H. Thielecke. Modelling environments in call-by-value programming languages. *Inform. and Comput.*, 185, 2003.
- [28] Sheng Liang, Paul Hudak, and Mark P. Jones. Monad transformers and modular interpreters. In *Proc. POPL 1995*, pages 333–343, 1995.
- [29] Bachuki Mesablishvili. Monads of effective descent type and comonadicity. *Theory and Applications of Categories*, 16(1):1–45, 2006.
- [30] F Métayer. State monads and their algebras. arXiv:math/0407251v1, 2004.
- [31] E. Moggi. Computational lambda-calculus and monads. In *Proceedings of the 4th Annual Symposium on Logic in Computer Science*, pages 14–23, Asilomar, CA, 1989. IEEE Computer Society Press.
- [32] P. W. O’Hearn and J. C. Reynolds. From Algol to polymorphic linear lambda-calculus. *J. ACM*, 47(1):167–223, 2000.
- [33] Pierre-Marie Pédrot. On the semantics of the effect calculus. Master’s thesis, ENS Lyon, 2010. Available at <http://perso.ens-lyon.fr/pierremarie.pedrot/reports/rapport-m1-hasegawa.pdf>.
- [34] G. Plotkin. Call-by-name, call-by-value, and the λ -calculus. *Theoret. Comp. Sci.*, 1:125–159, 1975.
- [35] G. Plotkin and J. Power. Tensors of comodels and models for operational semantics. In *Proc. MFPS XXIV*, volume 218 of *Electr. Notes Theor. Comput. Sci.*, pages 295–311. Elsevier, 2008.
- [36] G. Plotkin and M. Pretnar. Handlers of algebraic effects. In *Proc. ESOP'09*, volume 5502 of *LNCS*, pages 80–94. Springer, 2009.
- [37] G. D. Plotkin and J. Power. Notions of computation determine monads. In *Proc. FOSSACS'02*, volume 2620. Springer, 2002.
- [38] G. D. Plotkin and J. Power. Algebraic operations and generic effects. *Appl. Categ. Structures*, 11(1):69–94, 2003.
- [39] Gordon D Plotkin. Some varieties of equational logic. In *Essays dedicated to Joseph A. Goguen*, volume 4060 of *Lect. Notes in Comput. Sci.*, pages 150–156. Springer, 2006.
- [40] A. J. Power and O. Shkaravska. From comodels to coalgebras: State and arrays. In *Proc. CMCS'04*, volume 106 of *Electr. Notes Theor. Comput. Sci.*, pages 297–314. Elsevier, 2004.

- [41] John Power. Generic models for computational effects. *Theoret. Comput. Sci.*, 364(2):254–269, 2006.
- [42] John Power and Edmund Robinson. Premonoidal categories and notions of computation. *Math. Structures Comput. Sci.*, 7(5):453–468, 1997.
- [43] Matija Pretnar. *The logic and handling of algebraic effects*. PhD thesis, School of Informatics, University of Edinburgh, 2010.
- [44] Kurt Sieber. Full abstraction for the second order subset of an Algol-like language. In *Proc. MFCS*, pages 608–617, 1994.
- [45] Zoltan Somogyi, Fergus Henderson, and Thomas C. Conway. The execution algorithm of Mercury, an efficient purely declarative logic programming language. *J. Log. Program.*, 29(1-3):17–64, 1996.
- [46] C. Strachey. The varieties of programming language. In *Proc. International Computing Symposium*, pages 222–233. Cini Foundation, Venice, 1972. Also Tech. Monograph PRG-10, Univ. Oxford (1973).
- [47] Hayo Thielecke. *Categorical structure of continuation passing style*. PhD thesis, Univ. Edinburgh, 1997.

APPENDIX A. CATEGORIES OF MODELS

A.1. The 2-category \mathfrak{Cnr} of enriched models. We first define a notion of morphism of enriched call-by-value model and transformations between these. This gives a 2-category \mathfrak{Cnr} . Let (\mathbf{V}, \mathbf{C}) and $(\mathbf{V}', \mathbf{C}')$ be enriched call-by-value models (Def. 2.1). A morphism from (\mathbf{V}, \mathbf{C}) to $(\mathbf{V}', \mathbf{C}')$ is a triple (F, G, λ) such that $F: \mathbf{V} \rightarrow \mathbf{V}'$ and $G: \mathbf{C} \rightarrow \mathbf{C}'$ are functors, and λ is a natural family of isomorphisms

$$\lambda_{A, \underline{B}}: G(A \cdot \underline{B}) \cong F(A) \cdot G(\underline{B})$$

The following three conditions must be satisfied:

- F preserves products (up to isomorphism).
- The following two coherence diagrams commute

$$\begin{array}{ccc} G(1 \cdot \underline{B}) & \xrightarrow{\lambda} & F1 \cdot G\underline{B} \\ \cong \downarrow & & \downarrow ! \cdot \underline{S} \\ G\underline{B} & \xleftarrow{\cong} & 1 \cdot G\underline{B} \end{array} \quad (\text{A.1})$$

$$\begin{array}{ccccc} G((A \times B) \cdot \underline{C}) & \xrightarrow{\cong} & G(A \cdot B \cdot \underline{C}) & \xrightarrow{\lambda} & FA \cdot G(B \cdot \underline{C}) \\ \lambda \downarrow & & & & \downarrow FA \cdot \lambda \\ F(A \times B) \cdot G\underline{C} & \xrightarrow{\langle F\pi_1, F\pi_2 \rangle \cdot G\underline{C}} & (FA \times FB) \cdot G\underline{C} & \xrightarrow{\cong} & FA \cdot FB \cdot G\underline{C} \end{array} \quad (\text{A.2})$$

- The mate of λ^{-1} is an isomorphism $F(\mathbf{C}(\underline{B}, \underline{C})) \cong \mathbf{C}'(G\underline{B}, G\underline{C})$

Recall that the mate of λ^{-1} is the adjoint correspondent of

$$F(\mathbf{C}(\underline{B}, \underline{C})) \cdot G\underline{B} \xrightarrow{\lambda^{-1}} G(\mathbf{C}(\underline{B}, \underline{C}) \cdot \underline{B}) \xrightarrow{G(ev)} G(\underline{C})$$

where ev is the counit of the adjunction $(-) \cdot \underline{B} \dashv \mathbf{C}(\underline{B}, (-))$.

The 2-cells between morphisms $(F, G, \lambda), (F', G', \lambda') : (\mathbf{V}, \mathbf{C}) \rightarrow (\mathbf{V}', \mathbf{C}')$ are natural isomorphisms $\beta: F \cong F', \gamma: G \cong G'$ making the following diagram commute:

$$\begin{array}{ccc} G(A \cdot \underline{B}) & \xrightarrow{\lambda_{A,\underline{B}}} & FA \cdot G\underline{B} \\ \gamma \downarrow & & \downarrow \beta \cdot \gamma \\ G'(A \cdot \underline{B}) & \xrightarrow{\lambda'_{A,\underline{B}}} & F'A \cdot G'\underline{B} \end{array} \quad (\text{A.3})$$

The composition of 1-cells is defined as $(F', G', \lambda') \circ (F, G, \lambda) = (F'F, G'G, \lambda' \circ G'\lambda)$:

$$G'G(A \cdot \underline{B}) \xrightarrow{G'\lambda} G'(FA \cdot G\underline{B}) \xrightarrow{\lambda'} F'FA \cdot G'\underline{B}$$

The composition of 2-cells is simply pointwise $(\beta', \gamma') \circ (\beta, \gamma) = (\beta' \circ \beta, \gamma' \circ \gamma)$

A.2. The 2-category \mathfrak{Ecbv} . The objects of the 2-category \mathfrak{Ecbv} are tuples $(\mathbf{V}, \mathbf{C}, \underline{S})$ where (\mathbf{V}, \mathbf{C}) is an enriched call-by-value model and \underline{S} is a ‘state’ object in \mathbf{C} . A 1-cell $(\mathbf{V}, \mathbf{C}, \underline{S}) \rightarrow (\mathbf{V}', \mathbf{C}', \underline{S}')$ is a quadruple (F, G, λ, δ) such that $(F, G, \lambda): (\mathbf{V}, \mathbf{C}) \rightarrow (\mathbf{V}', \mathbf{C}')$ is a morphism of enriched call-by-value models and δ is an isomorphism

$$\delta: G\underline{S} \cong \underline{S}'$$

A 2-cell from (F, G, λ, δ) to $(F', G', \lambda', \delta')$ is a 2-cell of enriched call-by-value models (§A.1)

$$(\beta, \gamma): (F, G) \rightarrow (F', G')$$

such that

$$\begin{array}{ccc} G\underline{S} & \xrightarrow{\delta} & \underline{S}' \\ \gamma \downarrow & \nearrow \delta' & \\ G'\underline{S} & & \end{array} \quad (\text{A.4})$$

Composition of 1-cells and 2-cells in \mathfrak{Ecbv} is defined as in \mathfrak{Ent} . The state object isomorphisms are composed as follows:

$$G'G\underline{S} \xrightarrow{G'\delta} G'\underline{S}' \xrightarrow{\delta'} \underline{S}''.$$

A.3. The 2-category $\mathfrak{Kleisli}$. We introduce the 2-category $\mathfrak{Kleisli}$ whose objects are enriched Kleisli models (§3.1). Morphisms from $(\mathbf{V}, \mathbf{C}, J)$ to $(\mathbf{V}', \mathbf{C}', J')$ are morphisms of enriched models $(F, G, \lambda): (\mathbf{V}, \mathbf{C}) \rightarrow (\mathbf{V}', \mathbf{C}')$ such that $J' \circ F = G \circ J$. Note in particular that this means that F and G agree on objects, since J and J' are required to be identities on objects. A 2-cell $(F, G, \lambda) \rightarrow (F', G', \lambda')$ is a 2-cell of enriched models $(\beta, \gamma): (F, G) \rightarrow (F', G')$ such that $\gamma J = J' \beta: GJ \rightarrow J'F'$.

A.4. Sums. The category \mathfrak{dEnt} is defined to have distributive enriched models as objects, 1-cells as in \mathfrak{Ent} with the restriction that F and G must both preserve coproducts, and 2-cells as in \mathfrak{Ent} . The definitions of the 2-categories \mathfrak{Ecbv} and $\mathfrak{Kleisli}$ extend to 2-categories \mathfrak{dEcbv} and $\mathfrak{dKleisli}$ of distributive enriched models with state objects and distributive Kleisli models.

A.5. Effect theories. We now explain how to extend the 2-categories to models that support effect theories, expanding on Section 8.8. We first define what it means for a functor to preserve a value theory.

Definition A.1. Let \mathbf{V} and \mathbf{V}' be distributive categories with given interpretations of some fixed value theory (§8.5.1). A functor $F: \mathbf{V} \rightarrow \mathbf{V}'$ preserving products and coproducts *preserves the interpretation of the value theory* if $F[[\alpha]] = [[\alpha]]$ for all type constants α of the theory, and the diagram

$$\begin{array}{ccc} F([\bar{\alpha}]) & \xrightarrow{F[[f]]} & F([\beta]) \\ \langle F\pi_1, \dots, F\pi_n \rangle \downarrow & \nearrow [[f]] & \\ F[[\alpha_1]] \times \dots \times F[[\alpha_n]] & & \end{array}$$

commutes for all term constants $f: \bar{\alpha} \rightarrow \beta$ of the theory.

A.5.1. The 2-category \mathfrak{dEcbv}_E . For any effect theory E , the objects of the 2-category \mathfrak{dEcbv}_E are distributive enriched models with state and a chosen E -comodel structure on the state object (§8.5.3). Morphisms (1-cells) are morphisms (F, G, λ, δ) of \mathfrak{dEcbv} such that

- F preserves the value theory of E as in Definition A.1.
- The comodel structure is preserved, i.e., for each effect constant $e: \bar{\beta}; \bar{\alpha}_1 + \dots + \bar{\alpha}_n$ of E , the following diagram commutes

$$\begin{array}{ccc} G([\bar{\beta}] \cdot \underline{S}) & \xrightarrow{G[[e]]} & G([\bar{\alpha}_1] + \dots + [\bar{\alpha}_n] \cdot \underline{S}) \\ \downarrow & & \downarrow \\ [\bar{\beta}] \cdot \underline{S}' & \xrightarrow{[[e]]} & ([\bar{\alpha}_1] + \dots + [\bar{\alpha}_n]) \cdot \underline{S}' \end{array} \quad (\text{A.5})$$

where the vertical maps are constructed using λ, δ and the preservation of products and coproducts. (This makes sense because $F[[\alpha]] = [[\alpha]]$ for all type constants α .)

A 2-cell of \mathfrak{dEcbv}_E is a 2-cell $(\beta, \gamma): (F, G, \delta) \rightarrow (F', G', \delta')$ of \mathfrak{dEcbv} such that $\beta_{[[\alpha]]}$ is the identity for all type constants α of the theory E .

A.5.2. The 2-category $\mathfrak{dKleisli}_E$. For any effect theory E , the objects of the 2-category $\mathfrak{dKleisli}_E$ are distributive Kleisli models with given interpretations of E . Morphisms are morphisms of $\mathfrak{dKleisli}$ such that

- F preserves the value theory of E as in Definition A.1
- The interpretation of effect constants is preserved, i.e., for each $e: \bar{\beta}; \bar{\alpha}_1 + \dots + \bar{\alpha}_n$ of E , the following diagram commutes

$$\begin{array}{ccc} G([\bar{\beta}]) & \xrightarrow{G[[e]]} & G([\bar{\alpha}_1] + \dots + [\bar{\alpha}_n]) \\ \langle GJ(\pi_i) \rangle_i \downarrow & & \downarrow \\ [\bar{\beta}] & \xrightarrow{[[e]]} & ([\bar{\alpha}_1] + \dots + [\bar{\alpha}_n]) \end{array} \quad (\text{A.6})$$

where the left vertical map types because $G[[\beta_i]] = [[\beta_i]]$, and the right vertical map is constructed using the fact that G preserves coproducts.

The 2-cells of $\mathfrak{d}\mathfrak{R}\mathfrak{e}\mathfrak{i}\mathfrak{s}\mathfrak{i}_E$ are the 2-cells (β, γ) of $\mathfrak{d}\mathfrak{R}\mathfrak{e}\mathfrak{i}\mathfrak{s}\mathfrak{i}$ with $\beta_{[\alpha]}$ the identity for all type constants α of the theory E .

APPENDIX B. BI-INITIALITY OF SYNTACTIC MODELS

We sketch a proof of Theorem 2.4: the syntactic enriched model is bi-initial in the 2-category \mathfrak{Ecbv} of enriched models. Bi-initiality of the syntactic monad model (Theorem 3.3) can be proved in a similar manner.

Lemma B.1. *Suppose $(F, G, \delta): (\mathbf{V}, \mathbf{C}, \underline{S}) \rightarrow (\mathbf{V}', \mathbf{C}', \underline{S}')$ is a morphism in $\mathfrak{d}\mathfrak{Ecbv}_E$. Let $\llbracket - \rrbracket$ be the interpretation of ECBV in (\mathbf{V}, \mathbf{C}) , and let $\llbracket - \rrbracket'$ be the interpretation of ECBV in $(\mathbf{V}', \mathbf{C}')$ (as in §2.2, §8.5.3). The family of isomorphisms given by*

$$\begin{aligned} id: [\alpha]' &\rightarrow F[\alpha] \\ \delta^{-1}: [\underline{S}]' &= \underline{S}' \rightarrow G\underline{S} = G[\underline{S}] \end{aligned}$$

extends uniquely to a type-indexed family of isomorphisms $[\mathbf{A}]' \cong F([\mathbf{A}])$ and $[\mathbf{B}]' \cong G([\mathbf{B}])$ such that if $\Gamma \mid - \vdash t: \mathbf{A}$ and $\Gamma \mid \Delta \vdash u: \mathbf{B}$ then

$$\begin{array}{ccc} [\Gamma]' \xrightarrow{\llbracket t \rrbracket'} [\mathbf{A}]' & \text{and} & [\Gamma]' \cdot [\Delta]' \xrightarrow{\llbracket u \rrbracket'} [\mathbf{B}]' \\ \cong \downarrow & & \cong \downarrow \\ F[\Gamma] \xrightarrow{F\llbracket t \rrbracket} F[\mathbf{A}] & & G([\Gamma] \cdot [\Delta]) \xrightarrow{G\llbracket u \rrbracket} G[\mathbf{B}] \end{array} \quad (\text{B.1})$$

Moreover, if (β, γ) is a 2-cell then

$$\begin{array}{ccc} [\mathbf{A}]' \xrightarrow{\cong} F[\mathbf{A}] & \text{and} & [\mathbf{B}]' \xrightarrow{\cong} G[\mathbf{B}] \\ \searrow \cong & & \searrow \cong \\ & \downarrow \beta & \downarrow \gamma \\ & F'[\mathbf{A}] & G'[\mathbf{B}] \end{array} \quad (\text{B.2})$$

Proof notes. The isomorphisms are defined by induction on \mathbf{A} and \mathbf{B} . For example, in the case of $!\mathbf{A} \otimes \mathbf{B}$ we use the composite

$$[!\mathbf{A} \otimes \mathbf{B}]' = [\mathbf{A}]' \cdot [\mathbf{B}]' \xrightarrow{\cong} (F[\mathbf{A}]) \cdot (G[\mathbf{B}]) \xrightarrow{\lambda^{-1}} G([\mathbf{A}] \cdot [\mathbf{B}]) = G[!\mathbf{A} \otimes \mathbf{B}]$$

Commutativity of (B.1) and (B.2) is proved by induction on structure of terms and types respectively. We omit the lengthy verifications entirely, but remark that (A.1) is used to prove the case of linear variable introduction, and diagram (A.2) is used to prove the case of linear function application and the elimination rule for $!\mathbf{A} \otimes \mathbf{B}$. The case of effect constants is exactly the requirement (A.5).

Uniqueness is proved by induction on types. \square

Bi-initiality of the syntactic model (Theorem 2.4) follows from Lemma B.1 as follows. Given any other model, the unique morphism is given by interpretation of the syntax. To prove uniqueness, suppose (F, G, δ) is a 1-cell from $\mathfrak{d}\mathfrak{Ecbv}_E$ to $(\mathbf{V}, \mathbf{C}, \underline{S})$. Then Lemma B.1 gives the natural isomorphism: since interpretation in the syntactic model of terms with one variable is simply the identity, diagrams (A.1) prove naturality of the isomorphism. The required commutative diagrams (A.3) for 2-cells follow directly from definitions.

APPENDIX C. THE ADJUNCTION BETWEEN KLEISLI MODELS AND ENRICHED MODELS

C.1. **The 2-functor $\mathbf{St}: \mathfrak{Kleisli} \rightarrow \mathfrak{Ecbv}$.** We define the 2-functor $\mathbf{St}: \mathfrak{Kleisli} \rightarrow \mathfrak{Ecbv}$ by the following data:

$$\begin{aligned} \mathbf{St}(\mathbf{V}, \mathbf{C}, J) &\stackrel{\text{def}}{=} (\mathbf{V}, \mathbf{C}, 1) \\ \mathbf{St}(F, G) &\stackrel{\text{def}}{=} (F, G, J'(!)) \quad \text{for } (F, G): (\mathbf{V}, \mathbf{C}, J) \rightarrow (\mathbf{V}', \mathbf{C}', J') \\ \mathbf{St}(\beta, \gamma) &\stackrel{\text{def}}{=} (\beta, \gamma) \end{aligned}$$

Note that $J'(!)$ has the right type:

$$G(1) = G(J(1)) = J'(F(1)) \xrightarrow{J'(!)} J'(1) = 1.$$

C.2. **The 2-functor $\mathbf{Kl}: \mathfrak{Ecbv} \rightarrow \mathfrak{Kleisli}$.** Recall that \mathbf{Kl} is defined on objects in Section 5 as

$$\mathbf{Kl}(\mathbf{V}, \mathbf{C}, \underline{S}) \stackrel{\text{def}}{=} (\mathbf{V}, \mathbf{Kl}_{\underline{S}}, J_{\underline{S}})$$

where the category $\mathbf{Kl}_{\underline{S}}$ has the same objects as \mathbf{V} and homsets

$$\text{Hom}_{\mathbf{Kl}_{\underline{S}}}(A, B) \stackrel{\text{def}}{=} \text{Hom}_{\mathbf{C}}(A \cdot \underline{S}, B \cdot \underline{S})$$

On morphisms we define $\mathbf{Kl}(F, G, \lambda) \stackrel{\text{def}}{=} (F, \mathbf{Kl}(F, G), \lambda_{\mathbf{Kl}})$ where

$$\mathbf{Kl}(F, G): \mathbf{Kl}_{\underline{S}} \rightarrow \mathbf{Kl}_{\underline{S}'}$$

is the functor that maps an object A to FA and a morphism $f: A \cdot \underline{S} \rightarrow B \cdot \underline{S}$ to the following composite:

$$\begin{array}{ccc} FA \cdot \underline{S}' & \xrightarrow{A \cdot \delta} & FA \cdot G\underline{S} \xrightarrow{\lambda^{-1}} G(A \cdot \underline{S}) \\ \mathbf{Kl}(F, G)(f) \downarrow & & \downarrow G(f) \\ FB \cdot \underline{S}' & \xleftarrow{B \cdot \delta^{-1}} & FB \cdot G\underline{S} \xleftarrow{\lambda} G(B \cdot \underline{S}) \end{array} \quad (\text{C.1})$$

The natural transformation

$$\lambda_{\mathbf{Kl}}: \mathbf{Kl}(F, G)(A \cdot_{\mathbf{Kl}} B) \rightarrow F(A) \cdot_{\mathbf{Kl}} \mathbf{Kl}(F, G)(B)$$

has components given by

$$\langle F\pi_1, F\pi_2 \rangle \cdot \underline{S}': F(A \times B) \cdot \underline{S}' \rightarrow (F(A) \times F(B)) \cdot \underline{S}'$$

On 2-cells we define $\mathbf{Kl}(\beta, \gamma) = (\beta, \mathbf{Kl}(\beta, \gamma))$ where $\mathbf{Kl}(\beta, \gamma)$ is the natural transformation whose components are $\mathbf{Kl}(\beta, \gamma)_A = \beta_A \cdot \underline{S}': FA \cdot \underline{S}' \rightarrow F'A \cdot \underline{S}'$. Note that this defines a morphism in $\mathbf{Kl}_{\underline{S}'}$ from $\mathbf{Kl}(F, G)(A) = FA$ to $\mathbf{Kl}((F', G'))(A) = F'A$ as required.

C.3. The unit. If $(\mathbf{V}, \mathbf{C}, J)$ is a distributive Kleisli model, then

$$\mathbf{Kl}(\mathbf{St}(\mathbf{V}, \mathbf{C}, J)) = (\mathbf{V}, \mathbf{Kl}_1, J_1)$$

where \mathbf{Kl}_1 has the same objects as \mathbf{V} and $\text{Hom}_{\mathbf{Kl}_1}(A, B) = \text{Hom}_{\mathbf{C}}(A \times 1, B \times 1)$ and $J_1(A) = A$, $J_1(f) = J(f \times 1) = f \cdot 1$.

We define the unit $\eta_{(\mathbf{V}, \mathbf{C}, J)}$ as

$$\eta_{(\mathbf{V}, \mathbf{C}, J)} \stackrel{\text{def}}{=} (id, H, id): (\mathbf{V}, \mathbf{C}, J) \rightarrow (\mathbf{V}, \mathbf{Kl}_1, J_1)$$

where $H(A) = A$, and $H(f: A \rightarrow B)$ is the composition

$$A \times 1 \xrightarrow{J(\pi_1)} A \xrightarrow{f} B \xrightarrow{J(id, !)} B \times 1$$

The third component of η should be a natural transformation $H(A \cdot B) \rightarrow A \cdot_{\mathbf{Kl}} HB$. Since both sides are equal to $A \times B$ it makes sense to take the identity transformation.

Lemma C.1. *Each $\eta_{(\mathbf{V}, \mathbf{C}, J)}$ is an isomorphism in $\mathfrak{Kleisli}$.*

The next lemma states that the unit is a 2-natural transformation.

Lemma C.2. *For each pair of distributive Kleisli models $(\mathbf{V}, \mathbf{C}, J)$ and $(\mathbf{V}', \mathbf{C}', J')$, the following diagram of functors commutes.*

$$\begin{array}{ccc} \mathfrak{Kleisli}((\mathbf{V}, \mathbf{C}, J), (\mathbf{V}', \mathbf{C}', J')) & \xrightarrow{\mathbf{Kl} \circ \mathbf{St}} & \mathfrak{Kleisli}((\mathbf{V}, \mathbf{Kl}_1, J_1), (\mathbf{V}', \mathbf{Kl}'_1, J'_1)) \\ & \searrow (\eta_{(\mathbf{V}', \mathbf{C}', J')})^* & \downarrow (\eta_{(\mathbf{V}, \mathbf{C}, J)})^* \\ & & \mathfrak{Kleisli}((\mathbf{V}, \mathbf{C}, J), (\mathbf{V}', \mathbf{Kl}'_1, J'_1)) \end{array}$$

where f^* and g_* are the functors given by pre- and postcomposition by f and g respectively.

C.4. The counit. Let $(\mathbf{V}, \mathbf{C}, \underline{S})$ be a model of enriched call-by-value. Then

$$\mathbf{St}(\mathbf{Kl}(\mathbf{V}, \mathbf{C}, \underline{S})) = (\mathbf{V}, \mathbf{Kl}_{\underline{S}}, 1)$$

Recall that $\mathbf{Kl}_{\underline{S}}$ has the same objects as \mathbf{V} and set of morphisms defined as

$$\text{Hom}_{\mathbf{Kl}_{\underline{S}}}(A, B) = \text{Hom}_{\mathbf{C}}(A \cdot \underline{S}, B \cdot \underline{S})$$

Define the counit ϵ as

$$\epsilon_{(\mathbf{V}, \mathbf{C}, \underline{S})} \stackrel{\text{def}}{=} (id, I_{\underline{S}}, \delta): (\mathbf{V}, \mathbf{Kl}_{\underline{S}}, 1) \rightarrow (\mathbf{V}, \mathbf{C}, \underline{S})$$

where $I_{\underline{S}}$ is the functor defined by $I_{\underline{S}}(A) = A \cdot \underline{S}$, $I_{\underline{S}}(f) = f$, and δ is the isomorphism

$$\delta: I_{\underline{S}}(1) = 1 \cdot \underline{S} \rightarrow \underline{S}$$

The counit is not strictly natural, but it does satisfy the following naturality condition.

Lemma C.3. *Let $(\mathbf{V}, \mathbf{C}, \underline{S}), (\mathbf{V}', \mathbf{C}', \underline{S}')$ be two given objects of \mathfrak{Ecbv} . The following diagram commutes up to natural isomorphism.*

$$\begin{array}{ccc} \mathfrak{Ecbv}((\mathbf{V}, \mathbf{C}, \underline{S}), (\mathbf{V}', \mathbf{C}', \underline{S}')) & \xrightarrow{\mathbf{St} \circ \mathbf{Kl}} & \mathfrak{Ecbv}((\mathbf{V}, \mathbf{Kl}_{\underline{S}}, 1), (\mathbf{V}', \mathbf{Kl}'_{\underline{S}'}, 1)) \\ & \searrow (\epsilon_{(\mathbf{V}, \mathbf{C}, \underline{S})})^* \cong & \downarrow (\epsilon_{(\mathbf{V}', \mathbf{C}', \underline{S}')})^* \\ & & \mathfrak{Ecbv}((\mathbf{V}, \mathbf{Kl}_{\underline{S}}, 1), (\mathbf{V}', \mathbf{C}', \underline{S}')) \end{array}$$

C.5. **Triangle equalities.** One of the triangle equalities only holds up to isomorphism.

Proposition C.4. *Let $(\mathbf{V}, \mathbf{C}, \underline{S})$ be a distributive enriched model with state. The composite 1-cell*

$$\mathbf{Kl}(\epsilon_{(\mathbf{V}, \mathbf{C}, \underline{S})}) \circ \eta_{\mathbf{Kl}(\mathbf{V}, \mathbf{C}, \underline{S})}$$

is the identity on $\mathbf{Kl}(\mathbf{V}, \mathbf{C}, \underline{S})$.

Proposition C.5. *Let $(\mathbf{V}, \mathbf{C}, J)$ be a distributive Kleisli model. The composite 1-cell*

$$\epsilon_{\mathbf{St}(\mathbf{V}, \mathbf{C}, J)} \circ \mathbf{St}(\eta_{(\mathbf{V}, \mathbf{C}, J)})$$

is naturally isomorphic to the identity on $\mathbf{St}(\mathbf{V}, \mathbf{C}, J)$.

C.6. **Proof of Theorem 5.3.** We now prove Theorem 5.3: η and ϵ induce an equivalence of categories

$$\mathfrak{Ecbv}(\mathbf{St}(\mathbf{V}, \mathbf{C}, J), (\mathbf{V}', \mathbf{C}', \underline{S})) \simeq \mathfrak{Kleisli}((\mathbf{V}, \mathbf{C}, J), \mathbf{Kl}(\mathbf{V}', \mathbf{C}', \underline{S})).$$

In the following we use X to denote an object of \mathfrak{Ecbv} (rather than the much longer $(\mathbf{V}, \mathbf{C}, \underline{S})$) and Y to denote an object of $\mathfrak{Kleisli}$. The required equivalence of categories consists of the composite functors

$$\begin{aligned} \mathfrak{Ecbv}(\mathbf{St}(X), Y) &\xrightarrow{\mathbf{Kl}} \mathfrak{Kleisli}(\mathbf{Kl}(\mathbf{St}(X)), \mathbf{Kl}(Y)) \xrightarrow{(\eta_X)^*} \mathfrak{Kleisli}(X, \mathbf{Kl}(Y)) \\ \mathfrak{Kleisli}(X, \mathbf{Kl}(Y)) &\xrightarrow{\mathbf{St}} \mathfrak{Ecbv}(\mathbf{St}(X), \mathbf{St}(\mathbf{Kl}(Y))) \xrightarrow{(\epsilon_Y)^*} \mathfrak{Ecbv}(\mathbf{St}(X), Y) \end{aligned}$$

We need to prove that the two composites of these are naturally isomorphic to identities. For the first of the composites, consider the following sequences of identities and natural isomorphisms

$$\begin{aligned} (\epsilon_Y)_* \circ \mathbf{St} \circ \eta_X^* \circ \mathbf{Kl} &= (\epsilon_Y)_* \circ (\mathbf{St}(\eta_X))^* \circ \mathbf{St} \circ \mathbf{Kl} \\ &= (\mathbf{St}(\eta_X))^* \circ (\epsilon_Y)_* \circ \mathbf{St} \circ \mathbf{Kl} \\ &\cong (\mathbf{St}(\eta_X))^* \circ (\epsilon_{\mathbf{St}(Y)})^* && \text{Lemma C.3} \\ &= (\epsilon_{\mathbf{St}(Y)} \circ \mathbf{St}(\eta_X))^* \\ &\cong id^* && \text{Proposition C.5} \\ &= id \end{aligned}$$

The other composite is isomorphic to the identity for a similar reason. □