The enriched effect calculus: syntax and semantics

Egger, J., Møgelberg, R., & Simpson, A. (2012). The enriched effect calculus: syntax and semantics. Journal of Logic and Computation, doi: 10.1093/logcom/exs025

The Enriched Effect Calculus: Syntax and Semantics

Jeff Egger*
Department of Physics and Atmospheric Science
Dalhousie University, Halifax, N.S., Canada

Rasmus Ejlers Møgelberg†
IT University of Copenhagen
Copenhagen, Denmark.

Alex Simpson*
LFCS, School of Informatics,
University of Edinburgh, Scotland, UK.

February 9, 2012

Abstract

This paper introduces the *enriched effect calculus*, which extends established type theories for computational effects with primitives from linear logic. The new calculus provides a formalism for expressing linear aspects of computational effects; for example, the linear usage of imperative features such as state and/or continuations.

The enriched effect calculus is implemented as an extension of a basic *effect calculus* without linear primitives, which is closely related to Moggi's *computational metalanguage*, Filinski's *effect PCF* and Levy's *call-by-push-value*. We present syntactic results showing: the fidelity of the behaviour of the linear connectives of the enriched effect calculus; the conservativity of the enriched effect calculus over its non-linear core (the effect calculus); and the non-conservativity of intuitionistic linear logic when considered as an extension of the enriched effect calculus.

The second half of the paper investigates models for the enriched effect calculus, based on enriched category theory. We give several examples of such models, relating them to models of standard effect calculi (such as those based on monads), and to models of intuitionistic linear logic. We also prove soundness and completeness.

1 Introduction

The *computational metalanguage* was proposed by Moggi [30] as a general metalanguage for ascribing semantics to programming languages with effects, building on his own idea that computational effects can be encapsulated by the mathematical structure of a *strong monad* [29]. The metalanguage extends the simply-typed λ-calculus with a new type constructor T, where TA represents a type for computations that produce values of type A. Semantically, T is interpreted as a strong monad that captures the effects that computations may exhibit.

In [3], Benton and Wadler identify a close connection between Moggi's computational metalanguage and Girard's intuitionistic linear logic (ILL) [11]. They show that every model of ILL can be reconstrued as a model of the computational metalanguage, and this determines an interpretation of the computational metalanguage within ILL. However, the models of the

*Research supported by EPSRC research grant EP/F042043/1, which employed Egger as postdoctoral research fellow at the School of Informatics, University of Edinburgh, 2008–11.

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

computational metalanguage that arise from models of ILL are very special ones. Their monads are *commutative*. This means that the interpretation of the computational metalanguage in ILL validates an equation that is not always true for effects: the equation that asserts the insensitivity of computational effects to their order of execution.

Many computational effects are *not* commutative, for example: exceptions, state, input/output, and continuations. In [3, §8], Benton and Wadler write:

> *"We do not know if it is possible to define a non-commutative linear calculus which corresponds to a wider class of monad models."*

In this paper, we present such a calculus, the *enriched effect calculus*.

The enriched effect calculus can be viewed as an extension of Moggi's metalanguage with a judicious selection of type constructors from linear logic. We envisage that this calculus will be applicable to computational scenarios in which the manipulation of computational effects adheres to a discipline of linearity. For example, the usual *state monad* $S \to ((-) \times S)$ (where S is an object of *states*) has a linear counterpart $S \multimap (!(-) \otimes S)$, which accounts for the fact that state (unlike values) can neither be duplicated nor discarded, cf. [31]. Similarly, the *continuations monad* $((-) \to R) \to R$ (where R is a *result type*) has a linear version $((-) \to R) \multimap R$, which enforces the *linear usage* of continuations, a discipline that is ubiquitous amongst structured forms of control [4].

While examples such as the above can be formulated in ILL itself, we believe our enriched effect calculus to be the natural home for them. Indeed, the enriched effect calculus has two main advantages over ILL. First, it is weaker than ILL, and hence applicable more widely (models of ILL are a strict subset of models of the enriched effect calculus). Second, the tight connection between the enriched effect calculus and the standard monad-based calculi for effects, which we describe in this paper, makes the former a natural vehicle for formalising the general phenomenon of interactions between linearity and effects, including the *linear usage of effects* [14]. We shall outline several potential applications of the enriched effect calculus to such examples in this paper (see Examples 3.1–3.5 in Section 3). Detailed treatment of some of these examples appears in companion papers [7, 9, 28].

In this paper, our enriched effect calculus is defined as an extension of a basic *effect calculus*, which is presented in Section 2. The effect calculus is a simple extension of Moggi's computational metalanguage with a notion of *computation type*, as used in Filinski's *effect PCF* [10] and in Levy's *call-by-push-value (CBPV)* [21]. (Indeed, as we show in Appendix A, the version of our effect calculus with sums is equivalent to CBPV.)

The *enriched effect calculus* is defined, in Section 3, by extending the basic effect calculus with a selection of constructs from linear logic. In the enriched effect calculus, in contrast to linear logic, these constructs can only be used in certain restricted combinations (for example, the linear function space constructor $\multimap$ cannot be iterated). Nevertheless, the enriched effect calculus is expressive enough to formulate several computational situations in which linearity and effects interact; see Examples 3.1–3.5.

Section 4 surveys syntactic properties of the enriched effect calculus. First, it is shown that its various linear primitives enjoy the same interrelationships as in linear logic (Proposition 4.1). This, in part, justifies our choice of using notation and terminology from linear logic to describe these primitives. Second, we state two main syntactic theorems comparing the enriched effect calculus (EEC) with the basic (unenriched) effect calculus (EC). The enriched calculus EEC is a conservative extension of EC in two senses. Theorem 4.3 states that EEC is *syntactically conservative* over EC, meaning that every EEC term of EC type is equal (in EEC) to an EC term (more concisely, the embedding of EC in EEC is *full*). Theorem 4.4 states that EEC is *equationally conservative* over EC, meaning that any equation between EC terms that is

provable in EEC is already provable in EC (more concisely, the embedding is *faithful*). These two theorems give partial justification to our claim that EEC is an extension of effect calculi compatible with arbitrary computational effects. This brings us back to the quote from Benton and Wadler [3] above, which resulted from the general incompatibility of ILL with arbitrary effects. In our setting, this incompatibility of ILL manifests itself as the failure of conservativity of ILL over EEC (syntactic and equational conservativity both fail), as is shown at the end of Section 4.

The second half of the paper, Sections 5–7, is devoted to category-theoretic models of the calculi EC and EEC. As models of the basic effect calculus, we take (an appropriate version of) Levy's *adjunction models* [22], which are the natural models for calculi based on computation types. As models of the enriched effect calculus, we take adjunction models with the extra structure needed to model the linear connectives. In this paper, both notions of model are formulated in terms of *enriched category theory* [17], and Section 5 reviews the required background from this area. Section 6 defines the various notions of model we need, and provides several examples of models. In particular, it is shown how many models of computational effects based on monads give rise to models of EEC. This further addresses the Benton and Wadler [3] quote above. Finally, in Section 7, soundness and completeness are proved with respect to the models considered.

Two appendices are included. As already mentioned, Appendix A formulates the equivalence between our basic effect calculus with sums and Levy's CBPV. Appendix B provides the machinery (a confluent and normalizing rewrite relation) needed to prove our syntactic conservativity result, Theorem 4.3.

In the present paper, the equational conservativity result (Theorem 4.4) is stated without proof. Our proof of this works by showing that every model of EC has a (full) structure-preserving embedding into a model of EEC. In order to give a precise formulation of this result, it is necessary to develop the appropriate notion of morphism of models. As it turns out, a proper development of this notion requires significant extra category-theoretic machinery. Hence, in order both to keep the present paper to reasonable length, and not to swamp the presentation with technical category theory, we defer this result to a paper devoted entirely to the category-theoretic model theory of EC and EEC [8]. (The reader who is interested in an outline of this proof is referred to the conference version of the present paper [6].) We remark, that it is this semantic embedding theorem that answers Benton and Wadler's (implicit) question quoted above: the enriched effect calculus is compatible with *any* monad model, since any such can first be presented as an EC model and then fully embedded into an EEC model.

PART I: SYNTAX

2 A basic effect calculus

Moggi's *computational metalanguage*, [30], extends the simply-typed λ-calculus with new types TA, which type computations (possibly with effects) that produce values of type A. The new type has an associated "let" operator, which performs the *Kleisli extension* of a map $A \to TB$ to a map $TA \to TB$. This can be seen as a restricted form of elimination rule for the type TA. Filinski [10] generalises this elimination rule to apply to a wider class of "target" types than those of the form TB, and develops a calculus for this based on classifying such types as special *computation types* within a broader class of *value types*. Such a generalisation is useful for interpreting call-by-name languages. Its importance has been thoroughly established by Levy, whose *call-by-push-value (CBPV)* paradigm [21] is based on the distinction between computation and value types.

We define our *effect calculus* as a canonical calculus incorporating the above ideas. Following Moggi, we include a type constructor for computations. Following Filinski and Levy we classify types as value types and computation types. Because we have two classes of types, we assume two classes of type constants. We use $\alpha, \beta, \ldots$ to range over a set of *value type constants*, and $\underline{\alpha}, \underline{\beta}, \ldots$ to range over a disjoint set of *computation type constants*. We then use $\mathsf{A}, \mathsf{B}, \ldots$ to range over *value types*, and $\underline{\mathsf{A}}, \underline{\mathsf{B}}, \ldots$ to range over *computation types*, which are specified by the grammar below:

$$\mathsf{A} ::= \alpha \mid \underline{\alpha} \mid 1 \mid \mathsf{A} \times \mathsf{B} \mid \mathsf{A} \to \underline{\mathsf{B}} \mid \,!\mathsf{A} \quad \big[\mid 0 \mid \mathsf{A} + \mathsf{B}\big]$$
$$\underline{\mathsf{A}} ::= \underline{\alpha} \mid 1 \mid \underline{\mathsf{A}} \times \underline{\mathsf{B}} \mid \mathsf{A} \to \underline{\mathsf{B}} \mid \,!\mathsf{A}$$

Here, the parenthesised component of the grammar for value types represents the optional inclusion of finite-sum types (including the empty type 0) into the calculus. Note that the inclusion of value type sums also enlarges the collection of computation types. We refer to the calculus without sums as the plain *effect calculus* (EC). Otherwise, we explicitly say the *effect calculus with sums* (EC+). Observe that both calculi enjoy the property that every computation type is also a value type.

Our notation for type constructors is standard, except that we use the linear exponential notation $!\mathsf{A}$ for Moggi's monadic type TA. (The reasons for this nonstandard choice will transpire later.) We follow Filinski in making computation types a subclass of value types. Levy, in contrast, keeps computation types and value types separate. He has an operator F that turns a value type A into a computation type $F\mathsf{A}$, and conversely an operator U that maps a computation type $\underline{\mathsf{A}}$ to a value type $U\underline{\mathsf{A}}$. Levy's type $F\mathsf{A}$ corresponds to $!\mathsf{A}$ in our syntax, and his type $U\underline{\mathsf{A}}$ is simply $\underline{\mathsf{A}}$ itself. Two reasons for our choice of omitting U and subsuming computation types as value types are: the streamlined syntax leads to a very economical type system (see below) with no loss of information, since one can establish an equivalence between the two systems (see Appendix A); and the term syntax is not cluttered with (inferable) conversions between values and computations. A further point that deserves comment is that, as in Levy's CBPV, function types are restricted to those with computation-type codomain (i.e., those of the form $\mathsf{A} \to \underline{\mathsf{B}}$). This choice differs from the conference version of the paper [6], in which a full function space between value types was included in EC. The restriction on function types in the present version has been imposed in order to obtain the results of Appendix A, where it is shown that the calculus EC+ is equivalent to Levy's CBPV. This provides technical substantiation for the remarks about our streamlined syntax above.

A third benefit of our syntactic formulation of EC, which is specifically relevant to the goals of the present paper: our choice of syntax provides a transparent foundation for the extension with linear logic connectives in Section 3. In order to ease the transition to this linear calculus, we build a notion of linearity directly into the typing judgements of the basic effect calculus. This notion has an intuitive motivation. Following Levy [21], we view value types as typing *values*, which are static entities, and we view computation types as typing *computations*, which are dynamic entities. If a term t has computation type, and contains a parameter z of computation type then there is a natural notion of t depending linearly on z: the execution of the computation t contains within it exactly one execution of the computation z. Since computations may perform arbitrary effects including nontermination, such a linear dependency can only hold in general if the execution of z is the first subcomputation performed in the execution of t. (If, for example, a computation that diverges were due to be performed before z then z might never be executed.) Thus we may rephrase: t depends linearly on z if the execution of the computation t begins with the execution of the computation z. Accordingly, we have arrived at a notion of t depending linearly on z that is similar in spirit to saying that $t[z]$ is an evaluation context. The situation for value types is fundamentally different. Since

values are static, they are reasonably considered as pervasive entities that might be used any number of times. Accordingly, we do not build in any notion of a term t depending linearly on a parameter x of value type.

The above discussion is intended to give informal motivation for considering typing rules for the effect calculus based on two judgement forms:

$$\text{(i)}\ \ \Gamma \mid - \vdash t\colon \mathsf{A}$$
$$\text{(ii)}\ \ \Gamma \mid z\colon \underline{\mathsf{A}} \vdash t\colon \underline{\mathsf{B}}\ ,$$

where Γ is a context of value-type assignments to variables, i.e., a finite-domain partial function from variables to value types. On the right of Γ is a *stoup* (following the terminology of [13]), which may either be empty, as in the case of judgement (i), or may consist of a unique type assignment $z\colon \underline{\mathsf{A}}$, in which case the type on the right of the turnstyle is also required to be a computation type, as in (ii). The purpose of judgement (i) is merely to assert that the term t has value type A in (value) context Γ. Judgement (ii) asserts that t is a computation of type $\underline{\mathsf{B}}$ (in context Γ) which depends linearly on the computation z of type $\underline{\mathsf{A}}$. Note how these two judgement forms correspond to the informal discussion of linearity given above. This discussion also provides some intuitive motivation for the restriction of the linear context to a stoup containing at most one variable of computation type. Since a linearly-used parameter z (necessarily of computation type) must be executed first in the execution of t, it is natural that just one variable can enjoy this property.

The typing rules are presented in Figures 1–3. Figure 1 gives the rules for the pure effect calculus. Figure 2 allows terms built from a specified signature Σ of operations. Two kinds of operation may be declared in Σ. A *non-linear* operation f is given a type specification of the form $(\mathsf{A}_1, \ldots, \mathsf{A}_k) \to \mathsf{B}$, where $k \geq 0$ and $\mathsf{A}_1, \ldots, \mathsf{A}_k, \mathsf{B}$ are value types. The case $k = 0$ allows operations to include constants. A *linear* operation g is given a type specification of the form $(\mathsf{A}_1, \ldots, \mathsf{A}_k \mid \underline{\mathsf{B}}) \to \underline{\mathsf{C}}$, where $\underline{\mathsf{B}}$ and $\underline{\mathsf{C}}$ are computation types. Such an operation is declared linear in its last argument. A *signature* Σ is then a function from a given set of operation symbols to non-linear and linear type specifications. Finally, Figure 3 presents the additional typing rules needed to add sums to the effect calculus.

In the rules of Figures 1–3, Σ is a signature of operations, Γ ranges over contexts, and Δ ranges over an arbitrary (possibly empty) stoup. We use standard notation for the manipulation of signature and contexts. The rules are only considered applicable in the case of typing judgements that conform to (i) or (ii) above. Observe that the syntax of terms has been decorated with certain type annotations. These are included in order to obtain Proposition 2.1 below: uniqueness of types. Nevertheless, to save clutter, we shall normally omit type annotations from terms, writing, e.g., $?(t)$ instead of $?_{\mathsf{A}}(t)$, unless their presence is particularly helpful.

The positioning of the stoups Δ in the rules can be understood in terms of the intuitive definition of linearity given above. For example, the evaluation behaviour of the terms associated with $!A$ types can be understood following Moggi [29, 30]. In the introduction rule, the term $!t$ represents the trivial computation that immediately returns the value t of type A. There is no possibility in this of any linear dependency on a subcomputation z. In the elimination rule, the term $\mathsf{let}\ !x\ \mathsf{be}\ t\ \mathsf{in}\ u$ first evaluates the computation t, binds the result (if obtained) to x and then proceeds to evaluate the computation u. Clearly if z is evaluated first within t then it is also evaluated first within $\mathsf{let}\ !x\ \mathsf{be}\ t\ \mathsf{in}\ u$, and this justifies the positioning of Δ in the rule. Observe, however, that the following variation on the rule is not legitimised by our interpretation of linearity, and hence is not included in the calculus.

$$\frac{\Gamma \mid - \vdash t\colon !\mathsf{A} \qquad \Gamma,\, x\colon \mathsf{A} \mid \Delta \vdash u\colon \underline{\mathsf{B}}}{\Gamma \mid \Delta \vdash \mathsf{let}\ !x\ \mathsf{be}\ t\ \mathsf{in}\ u\colon \underline{\mathsf{B}}} \tag{1}$$

$$\frac{}{\Gamma,\, x\colon \mathsf{A} \mid - \vdash x\colon \mathsf{A}} \qquad \frac{}{\Gamma \mid z\colon \underline{\mathsf{A}} \vdash z\colon \underline{\mathsf{A}}} \qquad \frac{}{\Gamma \mid \Delta \vdash *\colon 1}$$

$$\frac{\Gamma \mid \Delta \vdash t\colon \mathsf{A} \quad \Gamma \mid \Delta \vdash u\colon \mathsf{B}}{\Gamma \mid \Delta \vdash \langle t, u\rangle\colon \mathsf{A} \times \mathsf{B}} \qquad \frac{\Gamma \mid \Delta \vdash t\colon \mathsf{A} \times \mathsf{B}}{\Gamma \mid \Delta \vdash \mathsf{fst}(t)\colon \mathsf{A}} \qquad \frac{\Gamma \mid \Delta \vdash t\colon \mathsf{A} \times \mathsf{B}}{\Gamma \mid \Delta \vdash \mathsf{snd}(t)\colon \mathsf{B}}$$

$$\frac{\Gamma,\, x\colon \mathsf{A} \mid \Delta \vdash t\colon \underline{\mathsf{B}}}{\Gamma \mid \Delta \vdash \lambda x\colon \mathsf{A}.\, t\colon \mathsf{A} \to \underline{\mathsf{B}}} \qquad \frac{\Gamma \mid \Delta \vdash s\colon \mathsf{A} \to \underline{\mathsf{B}} \quad \Gamma \mid - \vdash t\colon \mathsf{A}}{\Gamma \mid \Delta \vdash s(t)\colon \underline{\mathsf{B}}}$$

$$\frac{\Gamma \mid - \vdash t\colon \mathsf{A}}{\Gamma \mid - \vdash\, !t\colon\, !\mathsf{A}} \qquad \frac{\Gamma \mid \Delta \vdash t\colon\, !\mathsf{A} \quad \Gamma,\, x\colon \mathsf{A} \mid - \vdash u\colon \underline{\mathsf{B}}}{\Gamma \mid \Delta \vdash \mathsf{let}\, !x\, \mathsf{be}\, t\, \mathsf{in}\, u\colon \underline{\mathsf{B}}}$$

Figure 1: Typing rules for the effect calculus, EC

$$\frac{\Gamma \mid - \vdash t_1\colon \mathsf{A}_1 \quad \dots \quad \Gamma \mid - \vdash t_k\colon \mathsf{A}_k}{\Gamma \mid - \vdash f(t_1, \dots, t_k)\colon \mathsf{B}}\ f\colon (\mathsf{A}_1, \dots, \mathsf{A}_k) \to \mathsf{B} \in \Sigma$$

$$\frac{\Gamma \mid - \vdash t_1\colon \mathsf{A}_1 \quad \dots \quad \Gamma \mid - \vdash t_k\colon \mathsf{A}_k \quad \Gamma \mid \Delta \vdash u\colon \underline{\mathsf{B}}}{\Gamma \mid \Delta \vdash g(t_1, \dots, t_k \mid u)\colon \underline{\mathsf{C}}}\ g\colon (\mathsf{A}_1, \dots, \mathsf{A}_k \mid \underline{\mathsf{B}}) \to \underline{\mathsf{C}} \in \Sigma$$

Figure 2: Typing rules for a signature of operations

$$\frac{\Gamma \mid - \vdash t\colon 0}{\Gamma \mid \Delta \vdash ?_{\mathsf{A}}(t)\colon \mathsf{A}} \qquad \frac{\Gamma \mid - \vdash t\colon \mathsf{A}}{\Gamma \mid - \vdash \mathsf{inl}_{\mathsf{A},\mathsf{B}}(t)\colon \mathsf{A} + \mathsf{B}} \qquad \frac{\Gamma \mid - \vdash t\colon \mathsf{B}}{\Gamma \mid - \vdash \mathsf{inr}_{\mathsf{A},\mathsf{B}}(t)\colon \mathsf{A} + \mathsf{B}}$$

$$\frac{\Gamma \mid - \vdash s\colon \mathsf{A} + \mathsf{B} \quad \Gamma,\, x\colon \mathsf{A} \mid \Delta \vdash t\colon \mathsf{C} \quad \Gamma,\, y\colon \mathsf{B} \mid \Delta \vdash u\colon \mathsf{C}}{\Gamma \mid \Delta \vdash \mathsf{case}\, s\, \mathsf{of}\, (\mathsf{inl}(x).\, t; \mathsf{inr}(y).\, u)\colon \mathsf{C}}$$

Figure 3: Additional typing rules for the effect calculus with sums, EC+

The issue here is that any z in Δ is evaluated as part of u, and this occurs only after t has been evaluated first. Similar explanations can be given to the other rules. They rely on giving the products a lazy interpretation: components are only evaluated once projected out. (So, e.g., the linearity in the rule for 1 is correct because 1 is the empty product and $*$ can never be projected.)

As explained in detail in Appendix A, the typing rules for the effect calculus with sums are a concise reformulation of those for CBPV with *complex stacks*, as found in [21, 22]. Once again, our formulation has been chosen both for its economy and to make the extension with linear connectives transparent. Indeed, we have stayed close to linear logic notation (the main exception is the use of $\times$ for product rather than the usual linear &), and our typing rules are simply restrictions, from an arbitrary linear context to a stoup, of the rules for ILL in [1]. This, in part, motivates the nonstandard use of $!\mathsf{A}$ instead of $T\mathsf{A}$. The one mismatch here is the missing rule (1) above, which is valid in the context of ILL. In spite of this mismatch, it is our belief that the extension of the effect calculus with linear primitives presented below will make it clear that the overlap with linear logic is so strong that the linear notation is helpful more

$$\Gamma \mid \Delta \vdash t = *\colon 1 \qquad \text{if } \Gamma \mid \Delta \vdash t\colon 1$$
$$\Gamma \mid \Delta \vdash \mathsf{fst}(\langle t, u\rangle) = t\colon \mathsf{A} \qquad \text{if } \Gamma \mid \Delta \vdash t\colon \mathsf{A} \text{ and } \Gamma \mid \Delta \vdash u\colon \mathsf{B}$$
$$\Gamma \mid \Delta \vdash \mathsf{snd}(\langle t, u\rangle) = u\colon \mathsf{B} \qquad \text{if } \Gamma \mid \Delta \vdash t\colon \mathsf{A} \text{ and } \Gamma \mid \Delta \vdash u\colon \mathsf{B}$$
$$\Gamma \mid \Delta \vdash \langle \mathsf{fst}(t), \mathsf{snd}(t)\rangle = t\colon \mathsf{A} \times \mathsf{B} \qquad \text{if } \Gamma \mid \Delta \vdash t\colon \mathsf{A} \times \mathsf{B}$$
$$\Gamma \mid \Delta \vdash (\lambda x\colon \mathsf{A}.\, t)(u) = t[u/x]\colon \underline{\mathsf{B}} \qquad \text{if } \Gamma, x\colon \mathsf{A} \mid \Delta \vdash t\colon \underline{\mathsf{B}} \text{ and } \Gamma \mid - \vdash u\colon \mathsf{A}$$
$$\Gamma \mid \Delta \vdash \lambda x\colon \mathsf{A}.\, (t(x)) = t\colon \mathsf{A} \to \underline{\mathsf{B}} \qquad \text{if } \Gamma \mid \Delta \vdash t\colon \mathsf{A} \to \underline{\mathsf{B}} \text{ and } x \notin \Gamma, \Delta$$
$$\Gamma \mid - \vdash \mathsf{let}\ !x\ \mathsf{be}\ !t\ \mathsf{in}\ u = u[t/x]\colon \underline{\mathsf{B}} \qquad \text{if } \Gamma \mid - \vdash t\colon \mathsf{A} \text{ and } \Gamma,\, x\colon \mathsf{A} \mid - \vdash u\colon \underline{\mathsf{B}}$$
$$\Gamma \mid \Delta \vdash \mathsf{let}\ !x\ \mathsf{be}\ t\ \mathsf{in}\ u[!x/y] = u[t\,/\,y]\colon \underline{\mathsf{B}} \qquad \text{if } \Gamma \mid \Delta \vdash t\colon !\mathsf{A} \text{ and } \Gamma \mid y\colon !\mathsf{A} \vdash u\colon \underline{\mathsf{B}}$$

Figure 4: Equality axioms for the effect calculus

$$\Gamma \mid \Delta \vdash ?(t) = u[t/x]\colon \mathsf{A} \qquad \text{if } \Gamma \mid - \vdash t\colon 0 \text{ and } \Gamma, x\colon 0 \mid \Delta \vdash u\colon \mathsf{A}$$
$$\Gamma \mid \Delta \vdash \mathsf{case}\,\mathsf{inl}(t)\,\mathsf{of}\,(\mathsf{inl}(x).\, u; \mathsf{inr}(y).\, u') = u[t/x]\colon \mathsf{C} \qquad \text{if } \Gamma, x\colon \mathsf{A} \mid \Delta \vdash u\colon \mathsf{C} \text{ and } \Gamma, y\colon \mathsf{B} \mid \Delta \vdash u'\colon \mathsf{C} \text{ and } \Gamma \mid - \vdash t\colon \mathsf{A}$$
$$\Gamma \mid \Delta \vdash \mathsf{case}\,\mathsf{inr}(t)\,\mathsf{of}\,(\mathsf{inl}(x).\, u; \mathsf{inr}(y).\, u') = u'[t/x]\colon \mathsf{C} \qquad \text{if } \Gamma, x\colon \mathsf{A} \mid \Delta \vdash u\colon \mathsf{C} \text{ and } \Gamma, y\colon \mathsf{B} \mid \Delta \vdash u'\colon \mathsf{C} \text{ and } \Gamma \mid - \vdash t\colon \mathsf{B}$$
$$\Gamma \mid \Delta \vdash \mathsf{case}\,t\,\mathsf{of}\,(\mathsf{inl}(x).\, u[\mathsf{inl}(x)/z]; \mathsf{inr}(y).\, u[\mathsf{inr}(y)/z]) = u[t/z]\colon \mathsf{C} \qquad \text{if } \Gamma \mid \Delta \vdash t\colon \mathsf{A} + \mathsf{B} \text{ and } \Gamma, z\colon \mathsf{A} + \mathsf{B} \mid \Delta \vdash u\colon \mathsf{C}$$

Figure 5: Additional equality axioms for sums.

than it is misleading.

The next results state basic properties of the type systems. The straightforward proofs, by induction on derivations, are omitted. The results are stated once only, but apply, as written, to both EC and EC+ in the presence of an arbitrary signature Σ.

Proposition 2.1 (Uniqueness of types)**.** *If* $\Gamma \mid \Delta \vdash t\colon \mathsf{A}$ *and* $\Gamma \mid \Delta \vdash t\colon \mathsf{B}$ *then* $\mathsf{A} = \mathsf{B}$.

Proposition 2.2 (Weakening)**.** *If* $\Gamma \mid \Delta \vdash t\colon \mathsf{A}$ *and variable* x *does not appear in* Γ, Δ *then* $\Gamma,\, x\colon \mathsf{B} \mid \Delta \vdash t\colon \mathsf{A}$.

Proposition 2.3 (Substitution)**.**

1. *If* $\Gamma,\, x\colon \mathsf{A} \mid \Delta \vdash s\colon \mathsf{B}$ *and* $\Gamma \mid - \vdash t\colon \mathsf{A}$ *then* $\Gamma \mid \Delta \vdash s[t/x]\colon \mathsf{B}$.
2. *If* $\Gamma \mid x\colon \underline{\mathsf{A}} \vdash s\colon \underline{\mathsf{B}}$ *and* $\Gamma \mid \Delta \vdash t\colon \underline{\mathsf{A}}$ *then* $\Gamma \mid \Delta \vdash s[t/x]\colon \underline{\mathsf{B}}$.

Proposition 2.4 (Shift)**.** *If* $\Gamma \mid x\colon \underline{\mathsf{A}} \vdash t\colon \underline{\mathsf{B}}$ *then* $\Gamma,\, x\colon \underline{\mathsf{A}} \mid - \vdash t\colon \underline{\mathsf{B}}$.

Equational theories for the effect calculus are presented as sets of equations between well-typed terms in context. Each equation thus has the general form $\Gamma \mid \Delta \vdash t = u\colon \mathsf{A}$ where both t and u must be well typed, i.e., $\Gamma \mid \Delta \vdash t\colon \mathsf{A}$ and $\Gamma \mid \Delta \vdash u\colon \mathsf{A}$. The basic equalities, for the effect calculus EC, are presented in Figure 4. Additional equalities, for the calculus EC+ with sums, are given in Figure 5. In general, we consider either calculus over a given signature Σ of operations and an assumed set $\mathcal{E}$ of equations in context. The resulting equational theory

is then the smallest set of equations that contains $\mathcal{E}$, includes the axioms of Figures 4 and 5 (the latter in the case of EC+ only), and is closed under the expected (typed) congruence, α-equivalence and substitution rules. We write $\Gamma \mid \Delta \vdash_{\mathcal{E}} t = u \colon \mathsf{A}$ for the resulting equational theory (leaving the signature Σ implicit). Although we do not distinguish notationally between equational derivability in EC and in EC+, we will always make it clear from the context which is meant.

3 The enriched effect calculus

The enriched effect calculus is obtained by adding a selection of type constructors from linear logic to the effect calculus. As befits the setting, this needs to be done respecting both the distinction between value and computation types, and the interpretation of linearity as a concept related to the latter.

We start with linear function types. In our setting, we have a notion of linearity between computation types only. Thus we add a type $\underline{\mathsf{A}} \multimap \underline{\mathsf{B}}$, internalising the linear dependency of judgements $\Gamma \mid z \colon \underline{\mathsf{A}} \vdash t \colon \underline{\mathsf{B}}$. In order to have a calculus with a sufficiently wide collection of models (all monad models) it seems essential not to assume that $\underline{\mathsf{A}} \multimap \underline{\mathsf{B}}$ is a computation type in general. (Examples of models in which $\underline{\mathsf{A}} \multimap \underline{\mathsf{B}}$ cannot be a computation type will be discussed in Section 6.) We thus consider $\underline{\mathsf{A}} \multimap \underline{\mathsf{B}}$ as a value type only. This restriction fits in with the stoup-based typing judgements, since allowing a linear function to depend linearly on another parameter would naturally lead to typing rules involving multivariable linear contexts.

In linear logic, linear function space lies in an adjoint relationship with tensor product $\otimes$, which normally internalises the comma separating types in the linear context of a typing judgement. In our stoup-based system, there is at most one type in the linear context (the stoup), and so it seems awkward to implement the usual symmetric $\otimes$. Similarly, it is also difficult to find an appropriate $\otimes$ operation in a sufficiently general class of models. What does work, both syntactically and semantically, is an asymmetric version: for any value type A and computation type $\underline{\mathsf{B}}$, we include a new computation (and hence value) type $!\mathsf{A} \otimes \underline{\mathsf{B}}$. Note that this is the application of a single primitive binary constructor.[1] The hybrid notation is chosen to maintain consistency with linear logic.

Finally, we include: linear coproducts of computation types, $\underline{\mathsf{A}} \oplus \underline{\mathsf{B}}$ and $\underline{0}$, which are themselves computation types; and a full function space $\mathsf{A} \to \mathsf{B}$ between value types, which is itself a value type.

The resulting *enriched effect calculus* has types defined by extending the grammar for value and computation types of the effect calculus with the following additional type constructors.

$$\begin{aligned} \mathsf{A} &::= \ldots \mid \underline{\mathsf{A}} \multimap \underline{\mathsf{B}} \mid !\mathsf{A} \otimes \underline{\mathsf{B}} \mid \underline{0} \mid \underline{\mathsf{A}} \oplus \underline{\mathsf{B}} \mid \mathsf{A} \to \mathsf{B} \\ \underline{\mathsf{A}} &::= \ldots \mid !\mathsf{A} \otimes \underline{\mathsf{B}} \mid \underline{0} \mid \underline{\mathsf{A}} \oplus \underline{\mathsf{B}} \ . \end{aligned}$$

Here the ellipses mean that we include all the type constructs from the effect calculus. Once again, we are simultaneously defining two calculi: the *enriched effect calculus* EEC, and the *enriched effect calculus with sums* EEC+, where only the latter has the type constructs 0 and $\mathsf{A} + \mathsf{B}$. Notice that the new value-type constructor $\mathsf{A} \to \mathsf{B}$ of the enriched effect calculus subsumes the effect-calculus function space $\mathsf{A} \to \underline{\mathsf{B}}$, which can therefore be omitted from the full grammar for value types (but *not* from the grammar for computation types since $\mathsf{A} \to \mathsf{B}$ is a computation type only when B is too).

[1] We comment that a similar notion of asymmetric tensor has been introduced, in the context of game semantics, as a *sequoidal product*, by Laird and Churchill [19, 5].

$$\frac{\Gamma \,|\, z{:}\underline{\mathsf{A}} \vdash t{:}\,\underline{\mathsf{B}}}{\Gamma \,|\, - \vdash \lambda^{\circ} z{:}\underline{\mathsf{A}}.\, t{:}\,\underline{\mathsf{A}} \multimap \underline{\mathsf{B}}} \qquad \frac{\Gamma \,|\, - \vdash s{:}\,\underline{\mathsf{A}} \multimap \underline{\mathsf{B}} \quad \Gamma \,|\, \Delta \vdash t{:}\,\underline{\mathsf{A}}}{\Gamma \,|\, \Delta \vdash s[t]{:}\,\underline{\mathsf{B}}}$$

$$\frac{\Gamma \,|\, - \vdash t{:}\,\mathsf{A} \quad \Gamma \,|\, \Delta \vdash u{:}\,\underline{\mathsf{B}}}{\Gamma \,|\, \Delta \vdash\, !t \otimes u{:}\,!\mathsf{A} \otimes \underline{\mathsf{B}}} \qquad \frac{\Gamma \,|\, \Delta \vdash s{:}\,!\mathsf{A} \otimes \underline{\mathsf{B}} \quad \Gamma,\, x{:}\mathsf{A} \,|\, z{:}\underline{\mathsf{B}} \vdash t{:}\,\underline{\mathsf{C}}}{\Gamma \,|\, \Delta \vdash \mathsf{let}\, !x \otimes z\, \mathsf{be}\, s\, \mathsf{in}\, t{:}\,\underline{\mathsf{C}}}$$

$$\frac{\Gamma \,|\, \Delta \vdash t{:}\,\underline{0}}{\Gamma \,|\, \Delta \vdash \underline{?}(t){:}\,\underline{\mathsf{A}}} \qquad \frac{\Gamma \,|\, \Delta \vdash t{:}\,\underline{\mathsf{A}}}{\Gamma \,|\, \Delta \vdash \underline{\mathsf{inl}}(t){:}\,\underline{\mathsf{A}} \oplus \underline{\mathsf{B}}} \qquad \frac{\Gamma \,|\, \Delta \vdash t{:}\,\underline{\mathsf{B}}}{\Gamma \,|\, \Delta \vdash \underline{\mathsf{inr}}(t){:}\,\underline{\mathsf{A}} \oplus \underline{\mathsf{B}}}$$

$$\frac{\Gamma \,|\, \Delta \vdash s{:}\,\underline{\mathsf{A}} \oplus \underline{\mathsf{B}} \quad \Gamma \,|\, x{:}\underline{\mathsf{A}} \vdash t{:}\,\underline{\mathsf{C}} \quad \Gamma \,|\, y{:}\underline{\mathsf{B}} \vdash u{:}\,\underline{\mathsf{C}}}{\Gamma \,|\, \Delta \vdash \underline{\mathsf{case}}\, s\, \mathsf{of}\, (\underline{\mathsf{inl}}(x).\, t; \underline{\mathsf{inr}}(y).\, u){:}\,\underline{\mathsf{C}}}$$

$$\frac{\Gamma,\, x{:}\mathsf{A} \,|\, \Delta \vdash t{:}\,\mathsf{B}}{\Gamma \,|\, \Delta \vdash \lambda x{:}\mathsf{A}.\, t{:}\,\mathsf{A} \to \mathsf{B}} \qquad \frac{\Gamma \,|\, \Delta \vdash s{:}\,\mathsf{A} \to \mathsf{B} \quad \Gamma \,|\, - \vdash t{:}\,\mathsf{A}}{\Gamma \,|\, \Delta \vdash s(t){:}\,\mathsf{B}}$$

Figure 6: Additional typing rules for the enriched effect calculus.

$$\begin{array}{ll}
\Gamma \,|\, \Delta \vdash (\lambda^{\circ} x{:}\,\underline{\mathsf{A}}.\, t)[u] = t[u/x]{:}\,\underline{\mathsf{B}} & \text{if } \Gamma \,|\, x{:}\underline{\mathsf{A}} \vdash t{:}\,\underline{\mathsf{B}} \text{ and } \Gamma \,|\, \Delta \vdash u{:}\,\underline{\mathsf{A}} \\
\Gamma \,|\, - \vdash \lambda^{\circ} x{:}\,\underline{\mathsf{A}}.\, (t[x]) = t{:}\,\underline{\mathsf{A}} \multimap \underline{\mathsf{B}} & \text{if } \Gamma \,|\, - \vdash t{:}\,\underline{\mathsf{A}} \multimap \underline{\mathsf{B}} \text{ and } x \notin \Gamma \\
\Gamma \,|\, \Delta \vdash \mathsf{let}\, !x \otimes y\, \mathsf{be}\, !t \otimes s\, \mathsf{in}\, u = u[t,s/x,y]{:}\,\underline{\mathsf{C}} & \text{if } \Gamma \,|\, - \vdash t{:}\,\mathsf{A} \text{ and } \Gamma \,|\, \Delta \vdash s{:}\,\underline{\mathsf{B}} \\
 & \quad \text{and } \Gamma, x{:}\,\mathsf{A} \,|\, y{:}\,\underline{\mathsf{B}} \vdash u{:}\,\underline{\mathsf{C}} \\
\Gamma \,|\, \Delta \vdash \mathsf{let}\, !x \otimes y\, \mathsf{be}\, t\, \mathsf{in}\, u[!x \otimes y/z] = u[t/z]{:}\,\underline{\mathsf{C}} & \text{if } \Gamma \,|\, \Delta \vdash t{:}\,!\mathsf{A} \otimes \underline{\mathsf{B}} \text{ and } \Gamma \,|\, z{:}\,!\mathsf{A} \otimes \underline{\mathsf{B}} \vdash u{:}\,\underline{\mathsf{C}} \\
\Gamma \,|\, \Delta \vdash \underline{?}(t) = u[t/x]{:}\,\underline{\mathsf{A}} & \text{if } \Gamma \,|\, \Delta \vdash t{:}\,\underline{0} \text{ and } \Gamma \,|\, x{:}\,\underline{0} \vdash u{:}\,\underline{\mathsf{A}} \\
\Gamma \,|\, \Delta \vdash \underline{\mathsf{case}}\, \underline{\mathsf{inl}}(t)\, \mathsf{of}\, (\underline{\mathsf{inl}}(x).\, u; \underline{\mathsf{inr}}(y).\, u') & \text{if } \Gamma \,|\, x{:}\,\underline{\mathsf{A}} \vdash u{:}\,\underline{\mathsf{C}} \text{ and } \Gamma \,|\, y{:}\,\underline{\mathsf{B}} \vdash u'{:}\,\underline{\mathsf{C}} \\
\qquad = u[t/x] : \underline{\mathsf{C}} & \quad \text{and } \Gamma \,|\, \Delta \vdash t{:}\,\underline{\mathsf{A}} \\
\Gamma \,|\, \Delta \vdash \underline{\mathsf{case}}\, \underline{\mathsf{inr}}(t)\, \mathsf{of}\, (\underline{\mathsf{inl}}(x).\, u; \underline{\mathsf{inr}}(y).\, u') & \text{if } \Gamma \,|\, x{:}\,\underline{\mathsf{A}} \vdash u{:}\,\underline{\mathsf{C}} \text{ and } \Gamma \,|\, y{:}\,\underline{\mathsf{B}} \vdash u'{:}\,\underline{\mathsf{C}} \\
\qquad = u'[t/y] : \underline{\mathsf{C}} & \quad \text{and } \Gamma \,|\, \Delta \vdash t{:}\,\underline{\mathsf{B}} \\
\Gamma \,|\, \Delta \vdash \underline{\mathsf{case}}\, t\, \mathsf{of}\, (\underline{\mathsf{inl}}(x).\, u[\underline{\mathsf{inl}}(x)/z]; \underline{\mathsf{inr}}(y).\, u[\underline{\mathsf{inr}}(y)/z]) & \\
\qquad = u[t/z] : \underline{\mathsf{C}} & \text{if } \Gamma \,|\, \Delta \vdash t{:}\,\underline{\mathsf{A}} \oplus \underline{\mathsf{B}} \text{ and } \Gamma \,|\, z{:}\,\underline{\mathsf{A}} \oplus \underline{\mathsf{B}} \vdash u{:}\,\underline{\mathsf{C}} \\
\Gamma \,|\, \Delta \vdash (\lambda x{:}\,\mathsf{A}.\, t)(u) = t[u/x]{:}\,\mathsf{B} & \text{if } \Gamma, x{:}\mathsf{A} \,|\, \Delta \vdash t{:}\,\mathsf{B} \text{ and } \Gamma \,|\, - \vdash u{:}\,\mathsf{A} \\
\Gamma \,|\, \Delta \vdash \lambda x{:}\,\mathsf{A}.\, (t(x)) = t{:}\,\mathsf{A} \to \mathsf{B} & \text{if } \Gamma \,|\, \Delta \vdash t{:}\,\mathsf{A} \to \mathsf{B} \text{ and } x \notin \Gamma, \Delta
\end{array}$$

Figure 7: Additional equality rules for the enriched effect calculus.

The judgement forms for the enriched calculi are exactly as for the effect calculus (now using the extended range of types). The additional typing rules are presented in Figure 6. Again, they can be seen to be restrictions of standard intuitionistic linear logic rules, as in [1]. The basic syntactic properties of the typing relation, stated in Propositions 2.1–2.4, carry over to EEC and EEC+ verbatim. (There is thus no need to restate them.) The equality theory on terms is extended by the rules in Figure 7. As before, we write $\Gamma \,|\, \Delta \vdash_{\mathcal{E}} t = u{:}\,\mathsf{A}$ for derivability in equational theory $\mathcal{E}$, and we shall make it clear from the context the system (EC, EC+, EEC, EEC+) in which derivability is intended.

The restriction that $\underline{\mathsf{A}} \multimap \underline{\mathsf{B}}$ is a value type, and the lack of a symmetric tensor have consequences on expressivity that may, at first, seem drastic. An obvious limitation is that linear function space does not iterate: neither $\underline{\mathsf{A}} \multimap (\underline{\mathsf{B}} \multimap \underline{\mathsf{C}})$ nor $(\underline{\mathsf{A}} \multimap \underline{\mathsf{B}}) \multimap \underline{\mathsf{C}}$ are allowed. However, it is possible to interleave linear function space with full function space. For example, $\underline{\mathsf{A}} \multimap (\mathsf{B} \to \underline{\mathsf{C}})$ is a value type, and $(\underline{\mathsf{A}} \multimap \underline{\mathsf{B}}) \to \underline{\mathsf{C}}$ is a computation (and hence value) type. We end this section with a sequence of examples showing that the enriched effect calculus is expressive enough to capture several computationally relevant situations in which linearity combines with computational effects.

Example 3.1 (Linearly-used state)**.** For any computation type $\underline{S}$ of "states", the linearly-used-state monad, $\underline{S} \multimap (!(-) \otimes \underline{S})$, is implementable as a monad on value types. (Note that the use of the asymmetric tensor makes it essential that $\underline{S}$ is a computation type.) This monad plays a fundamental role because, perhaps surprisingly, every monad can be construed as a linearly-used-state monad, see Example 4.2. This phenomenon is studied in more detail in [28].

Example 3.2 (Linearly-used continuations)**.** For any computation type $\underline{R}$ of "results", the linearly-used-continuations monad, $((-) \to \underline{R}) \multimap \underline{R})$ is implementable as a monad on value types. (Again, it is essential for $\underline{R}$ to be a computation type.) This monad, in fact, underpins a linear-use-CPS translation from EEC to itself, which has been studied in detail in [7, 9].

Example 3.3 (First-class control)**.** For every computation type $\underline{\mathsf{A}}$, there is a canonical linear function in EEC

$$\alpha_{\underline{\mathsf{A}}} := \lambda^{\circ} x{:}\underline{\mathsf{A}}.\, \lambda k{:}\underline{\mathsf{A}} \multimap \underline{0}.\, k(x) : \underline{\mathsf{A}} \;\multimap\; (\underline{\mathsf{A}} \multimap \underline{0}) \to \underline{0} \;.$$

By adding to EEC a signature of operations $\alpha'_{\underline{\mathsf{A}}} \colon (\ \mid (\underline{\mathsf{A}} \multimap \underline{0}) \to \underline{0}) \to \underline{\mathsf{A}}$ (that is, each $\alpha'_{\underline{\mathsf{A}}}$ has a single linear argument), together with equations asserting that each induced function (where we overload the meaning of α')

$$\alpha'_{\underline{\mathsf{A}}} \colon ((\underline{\mathsf{A}} \multimap \underline{0}) \to \underline{0}) \;\multimap\; \underline{\mathsf{A}}$$

is inverse to $\alpha_{\underline{\mathsf{A}}}$, one obtains an extension of EEC with control operators, equivalent to that obtained by adding the CBPV control primitives of Levy [21]. Here, $\underline{0}$, is to be viewed as a generic result type, and the α' constants implement an explicit control operator, called "letstk" in [21]. In particular, the linear arrow in the type $\underline{\mathsf{A}} \multimap \underline{0}$ implements the property that the continuation supplied to "letstk" is a stack, that is, an evaluation context. This simple and intuitive linear typing for control operators was first studied in [25], within the context of a polymorphic version of EEC. An interesting feature of this application of EEC is that the analogous extension of ILL is equationally inconsistent. Thus, ILL does not allow a similar modelling of first-class control operators using $\multimap$ to model stacks.

Example 3.4. (Single-threaded nondeterminism) Plotkin and Power have argued that finite nondeterminism should be modelled by a free-binary-semilatice monad P [32, Example 2.4]. However, if construed as binary semilattices over a category with an object **Bool** of booleans, then the free algebra property induces maps $\Diamond \colon P(\mathbf{Bool}) \longrightarrow \mathbf{Bool}$ and $\Box \colon P(\mathbf{Bool}) \longrightarrow \mathbf{Bool}$, via the boolean semilattice structures $(\mathbf{Bool}, \vee)$ and $(\mathbf{Bool}, \wedge)$ respectively. These maps compute existential and universal quantifications over all nondeterministic branches — operations that are not computable under the usual "single-threaded" interpretation of nondeterminism, under which computation proceeds along just one nondeterministic branch. A possible approach to modelling such single-threadedness is to use the linear function space of EEC, asking for nondeterministic choice to be given as binary semilattice structures

$$\underline{\mathsf{A}} \times \underline{\mathsf{A}} \;\multimap\; \underline{\mathsf{A}} \;,$$

over computation types. The idea here is that the linearity of the choice operation, coupled with the fact that booleans **Bool** typically form a value type not a computation type, means that there are no analogues of the $\Box$ and $\Diamond$ operations. Intuitively, single-threadedness is ensured because a linear function acting on a product $\underline{\mathsf{A}} \times \underline{\mathsf{A}}$ has to first choose which component of the product to work on, as in Girard's intuitive explanation of how the linear product, " &", works [11]. To properly substantiate this idea for modelling single-threaded nondeterminism, it would be good to have a fully abstract model in which nondeterminism is modelled in the above way. This is an interesting direction for future research.

Example 3.5. (Polymorphism) It is possible to combine polymorphic types of the form $\forall X.\, \mathsf{A}$ and $\forall \underline{X}.\, \mathsf{A}$ with EEC+, where the quantifiers quantify over value type and computation types respectively. The resulting calculus supports a theory of relational parametricity for languages with computational effects. This is developed in [25, 26, 27], where it is shown (in effect) how full EEC+ arises from a basic type theory containing only universal quantifiers and (linear and non-linear) function types.

4 Properties of the Enriched Effect Calculus

Many of the familiar laws of linear logic transfer to the enriched effect calculus, insofar as they can be expressed. The proposition below states the most prominent of these, and also asserts laws of EEC+ specifically associated to value-type sums, which have no counterpart in linear logic. In the statement, we introduce the notation $\cong^\circ$ to denote a linear isomorphism between computation types.

Proposition 4.1. *The following isomorphisms hold in* EEC*:*

$$\mathsf{A} \to \underline{\mathsf{B}} \;\cong\; !\mathsf{A} \multimap \underline{\mathsf{B}} \tag{2}$$

$$(!\mathsf{A} \otimes \underline{\mathsf{B}}) \multimap \underline{\mathsf{C}} \;\cong\; \mathsf{A} \to (\underline{\mathsf{B}} \multimap \underline{\mathsf{C}}) \;\cong\; \underline{\mathsf{B}} \multimap (\mathsf{A} \to \underline{\mathsf{C}}) \tag{3}$$

$$\underline{\mathsf{A}} \multimap 1 \;\cong\; 1 \tag{4}$$

$$\underline{\mathsf{A}} \multimap (\underline{\mathsf{B}} \times \underline{\mathsf{C}}) \;\cong\; (\underline{\mathsf{A}} \multimap \underline{\mathsf{B}}) \times (\underline{\mathsf{A}} \multimap \underline{\mathsf{C}}) \tag{5}$$

$$\underline{0} \multimap \underline{\mathsf{A}} \;\cong\; 1 \tag{6}$$

$$(\underline{\mathsf{A}} \oplus \underline{\mathsf{B}}) \multimap \underline{\mathsf{C}} \;\cong\; (\underline{\mathsf{A}} \multimap \underline{\mathsf{C}}) \times (\underline{\mathsf{B}} \multimap \underline{\mathsf{C}}) \tag{7}$$

$$!\mathsf{A} \otimes !\mathsf{B} \;\cong^\circ\; !(\mathsf{A} \times \mathsf{B}) \tag{8}$$

$$!\mathsf{A} \otimes \underline{0} \;\cong^\circ\; \underline{0} \tag{9}$$

$$!\mathsf{A} \otimes (\underline{\mathsf{B}} \oplus \underline{\mathsf{C}}) \;\cong^\circ\; (!\mathsf{A} \otimes \underline{\mathsf{B}}) \oplus (!\mathsf{A} \otimes \underline{\mathsf{C}}) \tag{10}$$

$$!1 \otimes \underline{\mathsf{A}} \;\cong^\circ\; \underline{\mathsf{A}} \tag{11}$$

$$!(\mathsf{A} \times \mathsf{B}) \otimes \underline{\mathsf{C}} \;\cong^\circ\; !\mathsf{A} \otimes !\mathsf{B} \otimes \underline{\mathsf{C}} \tag{12}$$

The following isomorphisms hold in EEC+*:*

$$!0 \;\cong^\circ\; \underline{0} \tag{13}$$

$$!(\mathsf{A} + \mathsf{B}) \;\cong^\circ\; !\mathsf{A} \oplus !\mathsf{B} \tag{14}$$

$$!0 \otimes \underline{\mathsf{A}} \;\cong^\circ\; \underline{0} \tag{15}$$

$$!(\mathsf{A} + \mathsf{B}) \otimes \underline{\mathsf{C}} \;\cong^\circ\; (!\mathsf{A} \otimes \underline{\mathsf{C}}) \oplus (!\mathsf{B} \otimes \underline{\mathsf{C}}) \tag{16}$$

Proof. We provide the terms giving the isomorphisms for two cases: (8) and (12). The other cases are left to the reader, as is the verification of the mutual inverse property for each of the pairs of terms given below.

For (8), the terms giving the isomorphism are:

$$\lambda^\circ w\colon !\mathsf{A}\otimes !\mathsf{B}.\ \mathsf{let}\ !x\otimes z\ \mathsf{be}\ w\ \mathsf{in}\ \mathsf{let}\ !y\ \mathsf{be}\ z\ \mathsf{in}\ !\langle x,y\rangle\colon !\mathsf{A}\otimes !\mathsf{B}\multimap !(\mathsf{A}\times\mathsf{B})$$
$$\lambda^\circ w\colon !(\mathsf{A}\times\mathsf{B}).\ \mathsf{let}\ !z\ \mathsf{be}\ w\ \mathsf{in}\ !\mathsf{fst}(z)\otimes !\mathsf{snd}(z)\colon !(\mathsf{A}\times\mathsf{B})\multimap !\mathsf{A}\otimes !\mathsf{B}$$

For (12), the terms giving the isomorphism are:

$$\lambda^\circ v\colon !(\mathsf{A}\times\mathsf{B})\otimes\underline{\mathsf{C}}.\ \mathsf{let}\ !w\otimes z\ \mathsf{be}\ v\ \mathsf{in}\ !\mathsf{fst}(w)\otimes !\mathsf{snd}(w)\otimes z\colon !(\mathsf{A}\times\mathsf{B})\otimes\underline{\mathsf{C}}\multimap !\mathsf{A}\otimes !\mathsf{B}\otimes\underline{\mathsf{C}}$$
$$\lambda^\circ v\colon !\mathsf{A}\otimes !\mathsf{B}\otimes\underline{\mathsf{C}}.\ \mathsf{let}\ !x\otimes w\ \mathsf{be}\ v\ \mathsf{in}\ \mathsf{let}\ !y\otimes z\ \mathsf{be}\ w\ \mathsf{in}\ !\langle x,y\rangle\otimes z\colon !\mathsf{A}\otimes !\mathsf{B}\otimes\underline{\mathsf{C}}\multimap !(\mathsf{A}\times\mathsf{B})\otimes\underline{\mathsf{C}}$$

□

We make one comment on the above proposition. Due to our choice of notation, it may appear that law (12) is a special case of law (8). But this is not the case. Law (8) states properties of the $!\mathsf{A}\otimes\underline{\mathsf{C}}$ operator specialised to computation types $\underline{\mathsf{C}}$ of the form $!\mathsf{B}$, whereas law (12) applies to arbitrary $\underline{\mathsf{C}}$, but to value types A of the form $\mathsf{A}_1\times\mathsf{A}_2$. While one might consider such notational quirks as misleading, we in fact take the opposite view. The notation encourages one to transfer intuitions from linear logic to the connectives of EEC, and the proposition above underlines the broad sense in which such a transfer of intuitions is possible. Specifically, isomorphisms (2)–(12) above demonstrate that our linear connectives behave in the way that linear logic leads us to expect they should. Indeed, all the type isomorphisms of linear logic, of which we are aware, that can be formulated in our fragment, are valid as laws of EEC. (At the end of this section, we formulate a precise question based on this observation.) We take this as one justification for our decision to adopt linear logic notation, including the choice of replacing $T\mathsf{A}$ with $!\mathsf{A}$.

Example 4.2. An interesting example of an isomorphism derived from Proposition 4.1 is:

$$!\mathsf{A}\ \cong\ 1\to !\mathsf{A}\ \cong\ !1\multimap !\mathsf{A}\ \cong\ !1\multimap !(\mathsf{A}\times 1)\ \cong\ !1\multimap !\mathsf{A}\otimes !1\ .$$

Here, the second isomorphism is by (2), the fourth is by (8), and the other two are standard from typed λ-calculus. By defining $\underline{S}=!1$, we thus obtain:

$$!\mathsf{A}\ \cong\ \underline{S}\multimap !\mathsf{A}\otimes\underline{S}\ .$$

Furthermore, the strong monad structure (unit, multiplication and strength) over the operation $\mathsf{A}\mapsto !\mathsf{A}$ on computation types, transports along the isomorphism to the canonical linearly-used-state-monad structure on $\mathsf{A}\mapsto\underline{S}\multimap !\mathsf{A}\otimes\underline{S}$, cf. [31, 28]. This is the sense in which the claim that every monad is a linearly-used-state monad (see Example 3.1) is true.

We next state two main syntactic theorems about the enriched effect calculus. Taken together, these assert that the addition of the new linear type constructions is conservative over the basic effect calculus. In fact, more generally, for the four calculi, EC, EC+, EEC, EEC+, under consideration each larger calculus is conservative over each smaller one. Thus we let (X,Y) range over the following pairs of calculi, $(\mathrm{EC},\mathrm{EC}+)$, $(\mathrm{EC},\mathrm{EEC})$, $(\mathrm{EC},\mathrm{EEC}+)$, $(\mathrm{EC}+,\mathrm{EEC}+)$, $(\mathrm{EEC},\mathrm{EEC}+)$, and, in each case, we show that calculus Y is conservative over calculus X.

Theorem 4.3 (Syntactic conservativity)**.** *Let (X,Y) be as above. Suppose the signature Σ contains only types from the calculus X. If $\Gamma\mid\Delta\vdash u\colon\mathsf{A}$ in calculus Y, where Γ,Δ and A contain only types from calculus X, then there exists a term $\Gamma\mid\Delta\vdash t\colon\mathsf{A}$ typable in the calculus X such that $\Gamma\mid\Delta\vdash t=u\colon\mathsf{A}$ holds in calculus Y.*

This theorem is proved by a standard normalization argument. Every term u of EEC+ reduces to one t in a suitable normal form. If u is a system-Y term of system-X type, then its normal form t is a system-X term that is equal to u in the system-Y equational theory. The details, which are standard, can be found in Appendix B.

Theorem 4.4 (Equational conservativity)**.** *Let (X, Y) be as above. Suppose the signature Σ and equational theory $\mathcal{E}$ contain only types and terms from calculus X. If $\Gamma \mid \Delta \vdash s\colon \mathsf{A}$ and $\Gamma \mid \Delta \vdash t\colon \mathsf{A}$ are typable in calculus X, and $\Gamma \mid \Delta \vdash_{\mathcal{E}} s = t\colon \mathsf{A}$ holds in calculus Y, then $\Gamma \mid \Delta \vdash_{\mathcal{E}} s = t\colon \mathsf{A}$ holds in the equational theory of calculus X.*

A syntactic proof of Theorem 4.4 would have to show that all detours through non-system-X terms in a system-Y-derivation of $\Gamma \mid \Delta \vdash_{\mathcal{E}} s = t\colon \mathsf{A}$ are removable. We have been unable to obtain such a syntactic proof. Instead, the only proof we have of Theorem 4.4 is semantic. It works by relating category-theoretic models of the various calculi. In Part II of the paper below, we develop the basic notions of model used in the proof. The proof itself then works by showing that every model of system X fully embeds in a model of system Y. However, the notion of embedding here is somewhat subtle, and requires careful development of the correct notion of morphism between models. Because this is technically involved, we defer it to a companion paper devoted to the model theory of the various effect calculi [8]. Thus, in the present paper, we state Theorem 4.4 without proof. Nevertheless, a reasonably detailed outline of the argument can be found in the conference version of the present paper [6].

To end this section, we compare EEC with ILL. In contrast to the above results, we shall see that the embedding of EEC in ILL is neither syntactically nor equationally conservative. The first issue is how precisely to include EEC in ILL. Although we have been motivating EEC as a "fragment" of the latter, there is the discrepancy that ILL just has one kind of type, whereas EEC distinguishes between value and computation types. Accordingly, we start with a "crude" embedding of EEC in ILL, under which both value and computation type constants are interpreted as plain type constants in ILL, and all type constructors are interpreted by their evident (normally synonymous) linear counterparts. For this embedding, the counterexamples to equational and syntactic conservativity are essentially the same as those discussed in the paragraph preceding Remark 1 of [14]. Specifically, equational conservativity fails because ILL validates the "commutativity" equations

$$\mathsf{let}\ !x\ \mathsf{be}\ s\ \mathsf{in}\ \mathsf{let}\ !y\ \mathsf{be}\ t\ \mathsf{in}\ u\ =\ \mathsf{let}\ !y\ \mathsf{be}\ t\ \mathsf{in}\ \mathsf{let}\ !x\ \mathsf{be}\ s\ \mathsf{in}\ u\ , \tag{17}$$

where x, y are not free in s, t. Syntactic conservativity does not hold because, in ILL one has closed terms

$$\lambda x\colon !\mathsf{A}.\ \mathsf{let}\ !y\ \mathsf{be}\ x\ \mathsf{in}\ y\colon\ !\mathsf{A} \to \mathsf{A}\ ,$$

for all types A, whereas, for EEC, such terms are only available for computation types. (Of course, in ILL, one has similar terms of type $!\mathsf{A} \multimap \mathsf{A}$, but these do not violate syntactic conservativity since, in EEC, the linear function type exists only in the case that A is a computation type.)

The above immersion of EEC in ILL was called crude because a better comparison is to embed EEC in a version of ILL that makes a type distinction analogous to the EEC distinction between value and computation type. Such a version of ILL was proposed by Benton [2], whose *linear* types are analogous to our computation types, and whose *conventional* types are analogous to our value types. It is then natural to embed EEC in the resulting *linear non-linear* logic, by mapping value types to conventional types and computational types to linear types. Under such an embedding, the commutativity equation (17) above again demonstrates the failure of equational conservativity. Also, syntactic conservativity fails, but a different example is needed to show this. Consider the EEC term:

$$\lambda f\colon \mathsf{A} \to \underline{\mathsf{B}}.\ \lambda x\colon !\mathsf{A}.\ \mathsf{let}\ !y\ \mathsf{be}\ x\ \mathsf{in}\ f(y)\colon\ (\mathsf{A} \to \underline{\mathsf{B}}) \to (!\mathsf{A} \to \underline{\mathsf{B}})\ . \tag{18}$$

In EEC, it is not possible to strengthen this to a term

$$\lambda^{\circ} f\colon \mathsf{A} \to \underline{\mathsf{B}}.\ \lambda x\colon !\mathsf{A}.\ \mathsf{let}\ !y\ \mathsf{be}\ x\ \mathsf{in}\ f(y)\colon\ (\mathsf{A} \to \underline{\mathsf{B}}) \multimap (!\mathsf{A} \to \underline{\mathsf{B}})\ ,$$

because of the invalidity of rule (1) (see Section 2).[2] However, in linear non-linear logic, such a strengthened term does exist. It is worth noting, perhaps, that the counterexamples to both equational and syntactic conservativity share the same property that they arise because of the inability of linear logic to distinguish between the order of execution of computations.[3]

We end this section by formulating the question adumbrated in the paragraph following Proposition 4.1. Is it the case that, whenever a closed EEC term $t\colon \underline{\mathsf{A}} \multimap \underline{\mathsf{B}}$ or $t\colon \mathsf{A} \to \mathsf{B}$ is a (linear or non-linear respectively) isomorphism in ILL, it is also a (linear or non-linear) isomorphism in EEC?[4] A positive answer to this question would provide a systematic result that includes isomorphisms (2)–(12) of Proposition 4.1 as instances.

PART II: SEMANTICS

5 Preliminaries from enriched category theory

The name *enriched effect calculus* is partly an allusion to the central role that enriched category theory will play in its denotational models. In this section, we review some of the elements of enriched category theory, which will be needed in our treatment of models of all the calculi introduced so far (Section 6). The purpose of the present section is to collect definitions together with associated basic lemmata all in one place. The reader, who would prefer to follow the main narrative of the paper, may thus prefer to skip to Section 6 and to refer back to the present section as and when needed.

The central idea of enriched category theory is to replace the notion of *hom-set* with that of *hom-object*. In order for this to make sense, one must first choose a category, the *enriching category* $\mathbf{V}$, from which the hom-objects will be taken. The definitions require the enriching category $\mathbf{V}$ to be *monoidal*, that is, to have a specified monoidal product structure $(\otimes, I)$. (See [20] for a full definition of *monoidal category*.) Then a $(\mathbf{V}, \otimes, I)$-*category* (or, $\mathbf{V}$-*category*, if $\otimes$ and I are understood), $\mathbf{C}$, is given by a class of objects together with, for every pair of objects $\underline{A}, \underline{B}$, an associated $\mathbf{V}$-object $\mathbf{C}(\underline{A}, \underline{B})$, as well as $\mathbf{V}$-arrows

$$comp_{\underline{A},\underline{B},\underline{C}}\colon \mathbf{C}(\underline{B}, \underline{C}) \otimes \mathbf{C}(\underline{A}, \underline{B}) \longrightarrow \mathbf{C}(\underline{A}, \underline{C}) \tag{19}$$

$$\ulcorner id_{\underline{A}} \urcorner\colon I \longrightarrow \mathbf{C}(\underline{A}, \underline{A}) \tag{20}$$

providing composition and identities for $\mathbf{C}$. These are required to satisfy associativity and unitality axioms, which are rendered as commutative diagrams in $\mathbf{V}$. Similarly, a $(\mathbf{V}, \otimes, I)$-*functor* (or, $\mathbf{V}$-*functor*) $\mathbf{C} \longrightarrow \mathbf{D}$ consists of an assignment F of $\mathbf{C}$-objects to $\mathbf{D}$-objects together with a family of $\mathbf{V}$-morphisms

$$F_{\underline{A},\underline{B}}\colon \mathbf{C}(\underline{A}, \underline{B}) \longrightarrow \mathbf{D}(F\underline{A}, F\underline{B})$$

satisfying the usual functoriality axioms—now rendered as commutative diagrams in $\mathbf{V}$. A $\mathbf{V}$-functor F is called *fully faithful* if all of the $F_{\underline{A},\underline{B}}$ are isomorphisms. A $\mathbf{V}$-*natural transformation* from F to G consists of a family of $\mathbf{V}$-morphisms

$$\alpha_{\underline{A}}\colon I \longrightarrow \mathbf{D}(F\underline{A}, G\underline{A})$$

[2] For a semantic counterpart of this observation, let T be a monad on **Set** and consider the EEC model $F^T \dashv U^T\colon \mathbf{Set}^T \longrightarrow \mathbf{Set}$ of Proposition 6.8. Then, (18) denotes a function $U^T(\underline{B}^A) \to U^T(\underline{B}^{(U^T F^T A)})$. But, when the monad is non-commutative, this function is not in general the image under U^T of any homomorphism.

[3] Of course, linear logic was not designed for this purpose. What is surprising, perhaps, is that the linear primitives adapt so well to the context of EEC, in which such distinctions are significant.

[4] This can be reformulated semantically in terms of category-theoretic models of EEC and ILL, see Sections 6 and 7. Does the canonical morphism of models (as described in [8]) from the syntactic EEC model to the syntactic ILL model, which the foregoing discussion shows to be neither full nor faithful, nonetheless reflect isomorphisms?

whose naturality is, once again, rendered by a commutative diagram in $\mathbf{V}$. A $\mathbf{V}$-*adjunction* $F \dashv U$ between $(\mathbf{V}, \otimes, I)$-functors $F\colon \mathbf{D} \longrightarrow \mathbf{C}$ and $U\colon \mathbf{C} \longrightarrow \mathbf{D}$ can be equivalently defined either in terms of a pair of $\mathbf{V}$-natural transformations $\eta\colon \mathrm{Id}_{\mathbf{D}} \longrightarrow UF$, $\varepsilon\colon FU \longrightarrow \mathrm{Id}_{\mathbf{C}}$ satisfying the usual *triangle identities*, or in terms of $\mathbf{V}$-isomorphisms

$$\rho_{\underline{A},\underline{B}}\colon \mathbf{C}(F\underline{A}, \underline{B}) \longrightarrow \mathbf{D}(\underline{A}, U\underline{B}) \tag{21}$$

satisfying appropriate coherence diagrams in $\mathbf{V}$.[5] We refer readers to Kelly's book [17] for full definitions.

Enriched category theory generalises ordinary category theory in the sense that a $(\mathbf{Set}, \times, 1)$-category (-functor, -natural transformation, -adjunction) is just an ordinary locally small category (resp., functor, natural transformation, adjunction). Moreover, if $\mathbf{C}$ is a $\mathbf{V}$-category, one can form an ordinary category called the *underlying category* of $\mathbf{C}$; this has the same class of objects as $\mathbf{C}$ but its arrows are given by $\mathbf{V}$-morphisms of the form $f\colon I \longrightarrow \mathbf{C}(\underline{A}, \underline{B})$. Similarly, every $\mathbf{V}$-functor (-natural transformation, -adjunction) has an underlying functor (resp., natural transformation, adjunction).

This procedure of extracting a $\mathbf{Set}$-category from a $\mathbf{V}$-category is merely one case of a general result which will be useful in the sequel. Let us recall that a *monoidal functor* (or, *lax monoidal functor*, according to some authors) from $(\mathbf{V}, \otimes, I)$ to $(\mathbf{V}', \otimes', I')$ consists of a functor $M\colon \mathbf{V} \longrightarrow \mathbf{V}'$ together with an arrow $\eta\colon I' \longrightarrow M(I)$ and a natural transformation

$$\mu_{A,B}\colon MA \otimes' MB \longrightarrow M(A \otimes B)$$

satisfying appropriate associativity and unitality axioms. It is said to be *strong* if η and $\mu_{A,B}$ are isomorphisms. There is a natural notion of *monoidal natural transformation* between monoidal functors [20].

Lemma 5.1 ([17, p.3])**.** *Let $(M, \mu, \eta)\colon (\mathbf{V}, \otimes, I) \longrightarrow (\mathbf{V}', \otimes', I')$ be a monoidal functor. Application of M to hom-objects defines a 2-functor ${}_M(\cdot)$ from the 2-category of $\mathbf{V}$-categories to that of $\mathbf{V}'$-categories; that is, ${}_M\mathbf{C}$ has the same objects as $\mathbf{C}$ but hom-objects defined by ${}_M\mathbf{C}(\underline{A}, \underline{B}) = M(\mathbf{C}(\underline{A}, \underline{B}))$.*

Example 5.2. For any locally small monoidal category $(\mathbf{V}, \otimes, I)$, let $U\colon \mathbf{V} \longrightarrow \mathbf{Set}$ denote the functor $U = \mathbf{V}(I, -)$. Then U, together with the arrow $\ulcorner id_I \urcorner\colon 1 \longrightarrow \mathbf{V}(I, I) = UI$, and the natural transformation

$$UA \times UB = \mathbf{V}(I, A) \times \mathbf{V}(I, B) \xrightarrow{\otimes} \mathbf{V}(I \otimes I, A \otimes B) \xrightarrow{\sim} \mathbf{V}(I, A \otimes B) = U(A \otimes B),$$

form a monoidal functor $(\mathbf{V}, \otimes, I) \longrightarrow (\mathbf{Set}, \times, 1)$. Moreover, the corresponding 2-functor $\mathbf{C} \mapsto {}_U\mathbf{C}$ maps a $\mathbf{V}$-category to its underlying category.

Henceforth, we shall find it convenient to restrict our attention to enriching categories whose monoidal structure is both *symmetric* and *closed*. We write $[A \to (\cdot)]$ for the right adjoint to $(\cdot) \otimes A$. The assumption of closedness, means that $\mathbf{V}$ admits a canonical *self-enrichment*, $\mathbf{V}_{\mathrm{self}}$, with hom-objects as follows.[6]

$$\mathbf{V}_{\mathrm{self}}(A, B) = [A \to B] \ .$$

In practice, it is often convenient to elide the distinction between $\mathbf{V}$ and $\mathbf{V}_{\mathrm{self}}$, as this rarely causes confusion. For instance, if we speak of a $\mathbf{V}$-functor $\mathbf{C} \longrightarrow \mathbf{V}$, this can only make sense if $\mathbf{C}$ and $\mathbf{V}$ are $\mathbf{V}$-categories; so, in this case, we really mean $\mathbf{V}_{\mathrm{self}}$.

[5] In the case where $\mathbf{V}$ is symmetric and closed (see below), these diagrams are equivalent to the $\mathbf{V}$-naturality in $\underline{A}$ and $\underline{B}$ of (21).

[6] More generally, self-enrichedness applies to arbitrary monoidal $(\mathbf{V}, \otimes, I)$ for which every functor $(\cdot) \otimes A$ has a right adjoint, irrespective of symmetry.

Lemma 5.3. *Let $(\mathbf{V}, \otimes, I)$ and $(\mathbf{W}, \otimes, I)$ be closed symmetric monoidal categories and let $(M, \mu, \eta)\colon (\mathbf{W}, \otimes, I) \longrightarrow (\mathbf{V}, \otimes, I)$ be a monoidal functor. Then M admits a canonical enrichment as a $\mathbf{V}$-functor $M_{\mathrm{can}}\colon {}_M\mathbf{W} \to \mathbf{V}$.*

Proof. The Curry of

$$M[A \to B] \otimes MA \xrightarrow{\mu_{[A\to B],A}} M([A \to B] \otimes A) \xrightarrow{M(ev)} MB$$

defines the requisite arrow

$${}_M\mathbf{W}(A, B) = M[A \to B] \longrightarrow [MA \to MB] = \mathbf{V}(MA, MB)$$

□

Given a $\mathbf{V}$-object A, we say that a $\mathbf{V}$-category $\mathbf{C}$ has *A-fold copowers* (Kelly writes *tensors indexed by A*) if, for each object $\underline{B}$ in $\mathbf{C}$, there exists an object $A \cdot \underline{B}$ of $\mathbf{C}$ together with $\mathbf{V}$-isomorphisms

$$\psi_{A,\underline{B},\underline{C}}\colon \mathbf{C}(A \cdot \underline{B}, \underline{C}) \longrightarrow [A \to \mathbf{C}(\underline{B}, \underline{C})] \tag{22}$$

which are $\mathbf{V}$-natural in $\underline{C}$. The dual property is that of having *A-fold powers* (Kelly writes *cotensors indexed by A*): for each object $\underline{B}$ of $\mathbf{C}$, there must exist an object $\underline{B}^A$ of $\mathbf{C}$ and $\mathbf{V}$-isomorphisms

$$\xi_{A,\underline{B},\underline{C}}\colon \mathbf{C}(\underline{C}, \underline{B}^A) \longrightarrow [A \to \mathbf{C}(\underline{C}, \underline{B})] \tag{23}$$

again $\mathbf{V}$-natural in $\underline{C}$. The category $\mathbf{V}$ automatically has all A-fold powers and copowers, under its self-enrichment $\mathbf{V}_{\mathrm{self}}$. These are given by $A \cdot B = A \otimes B$ and $B^A = [A \to B]$ (and we shall henceforth often write B^A for $[A \to B]$).

Lemma 5.4. *Let $(\mathbf{V}, \otimes, I)$ and $(\mathbf{W}, \otimes, I)$ be closed symmetric monoidal categories, and let $(M, \mu, \eta)\colon (\mathbf{W}, \otimes, I) \longrightarrow (\mathbf{V}, \otimes, I)$ be a monoidal functor such that $M_{\mathrm{can}}\colon {}_M\mathbf{W} \longrightarrow \mathbf{V}$ has a left $\mathbf{V}$-adjoint, $L\colon \mathbf{V} \longrightarrow {}_M\mathbf{W}$. For every $\mathbf{W}$-category $\mathbf{D}$, and every object A of $\mathbf{V}$, if $\mathbf{D}$ has LA-fold (co)powers then ${}_M\mathbf{D}$ has A-fold (co)powers.*

Proof. Under the given hypotheses, for every object $\underline{B}$ of $\mathbf{D}$, we have isomorphisms

$$\begin{aligned}
[A \to {}_M\mathbf{D}(\underline{C}, \underline{B})] &= \mathbf{V}(A, M\mathbf{D}(\underline{C}, \underline{B})) \\
&= \mathbf{V}(A, M_{\mathrm{can}}\mathbf{D}(\underline{C}, \underline{B})) \\
&\cong {}_M\mathbf{W}(LA, \mathbf{D}(\underline{C}, \underline{B})) \\
&= M[LA \to \mathbf{D}(\underline{C}, \underline{B})] \\
&\cong M\mathbf{D}(\underline{C}, \underline{B}^{LA}) \\
&= {}_M\mathbf{D}(\underline{C}, \underline{B}^{LA})
\end{aligned}$$

$\mathbf{V}$-natural in $\underline{C}$. Hence $\underline{B}^{LA}$, as computed in $\mathbf{D}$, enjoys the correct universal property (23) to be $\underline{B}^A$, as computed in ${}_M\mathbf{D}$. By essentially the same argument, $LA \cdot \underline{B}$, as computed in $\mathbf{D}$, enjoys the correct universal property (22) to be $A \cdot \underline{B}$, as computed in ${}_M\mathbf{D}$. □

Now suppose that our enriching category, $\mathbf{V}$, also has finite products. Then $\mathbf{C}$ is said to have *finite $\mathbf{V}$-coproducts* if it has an object 0 and, for each pair of objects $\underline{A}, \underline{B}$ an object $\underline{A} + \underline{B}$, together with isomorphisms

$$\begin{aligned}
[\cdot]_{\underline{C}}\colon 1 &\cong \mathbf{C}(0, \underline{C}) \\
[-,-]_{\underline{A},\underline{B},\underline{C}}\colon \mathbf{C}(\underline{A}, \underline{C}) \times \mathbf{C}(\underline{B}, \underline{C}) &\cong \mathbf{C}(\underline{A} + \underline{B}, \underline{C})
\end{aligned}$$

$\mathbf{V}$-natural in $\underline{C}$. Dually, $\mathbf{C}$ is said to have *finite* $\mathbf{V}$*-products* if it has an object 1 and, for each pair of objects $\underline{A}, \underline{B}$ an object $\underline{A} \times \underline{B}$, together with isomorphisms

$$\langle\cdot\rangle_{\underline{C}} \colon 1 \cong \mathbf{C}(\underline{C}, 1)$$
$$\langle -,- \rangle_{\underline{A},\underline{B},\underline{C}} \colon \mathbf{C}(\underline{C},\underline{A}) \times \mathbf{C}(\underline{C},\underline{B}) \cong \mathbf{C}(\underline{C}, \underline{A} \times \underline{B})$$

$\mathbf{V}$-natural in $\underline{C}$.

It follows from naturality that the inverses of $[-,-]$ and $\langle -,-\rangle$ must be given by $\mathbf{V}$-morphisms

$$\ulcorner inl \urcorner \colon I \longrightarrow \mathbf{C}(\underline{A}, \underline{A} + \underline{B}) \qquad \ulcorner \pi_1 \urcorner \colon I \longrightarrow \mathbf{C}(\underline{A} \times \underline{B}, \underline{A})$$
$$\ulcorner inr \urcorner \colon I \longrightarrow \mathbf{C}(\underline{B}, \underline{A} + \underline{B}) \qquad \ulcorner \pi_2 \urcorner \colon I \longrightarrow \mathbf{C}(\underline{A} \times \underline{B}, \underline{B})$$

corresponding to arrows $inl \colon \underline{A} \longrightarrow \underline{A} + \underline{B}$, $inr \colon \underline{B} \longrightarrow \underline{A} + \underline{B}$, $\pi_1 \colon \underline{A} \times \underline{B} \longrightarrow \underline{A}$ and $\pi_2 \colon \underline{A} \times \underline{B} \longrightarrow \underline{B}$ in the underlying category of $\mathbf{C}$. These satisfy the usual universal properties. Similarly, there are unique $\mathbf{V}$-arrows $\ulcorner ? \urcorner \colon I \longrightarrow \mathbf{C}(0, \underline{C})$, $\ulcorner ! \urcorner \colon I \longrightarrow \mathbf{C}(\underline{C}, 1)$ corresponding to unique arrows $? : 0 \longrightarrow \underline{C}$, $! : \underline{C} \longrightarrow 1$ in the underlying category of $\mathbf{C}$. Hence, if $\mathbf{C}$ has finite $\mathbf{V}$-(co)products, then its underlying category, ${}_U\mathbf{C}$, has ordinary finite (co)products. Once again, this is part of a more general pattern.

Lemma 5.5. *If $\mathbf{C}$ has finite $\mathbf{V}$-(co)products and $(M, \mu, \eta) \colon (\mathbf{V}, \otimes, I) \longrightarrow (\mathbf{V}', \otimes', I')$ is a monoidal functor such that M also preserves finite products, then ${}_M\mathbf{C}$ has finite $\mathbf{V}'$-(co)products.*

Another useful lemma is the following.

Lemma 5.6 ([17, p. 50])**.** *If $\mathbf{C}$ has A-fold powers for every $\mathbf{V}$-object A, and ${}_U\mathbf{C}$ has (ordinary) finite coproducts, then $\mathbf{C}$ has finite $\mathbf{V}$-coproducts. Dually, if $\mathbf{C}$ has A-fold copowers for every $\mathbf{V}$-object A, and ${}_U\mathbf{C}$ has (ordinary) finite products, then $\mathbf{C}$ has finite $\mathbf{V}$-products..*

Finally, note that, for any small category $\mathbf{V}$, the presheaf category $\widehat{\mathbf{V}} = \mathbf{Set}^{\mathbf{V}^{\mathrm{op}}}$ is cartesian closed, and therefore self-enriched. Since the Yoneda functor $\mathbf{y}$ fully embeds $\mathbf{V}$ into $\widehat{\mathbf{V}}$, the category $\mathbf{V}$ inherits a $\widehat{\mathbf{V}}$-enriched structure with hom-objects $[\mathbf{y}A \to \mathbf{y}B]$. If $\mathbf{V}$ has products, then $\mathbf{y}$ preserves them, and one can use this to derive an alternative (isomorphic) description of this $\widehat{\mathbf{V}}$-category structure:

$$\mathbf{V}_{\mathrm{psh}}(A, B)(C) \;\cong\; \mathbf{V}(A \times C, B). \tag{24}$$

Again, we elide the distinction between $\mathbf{V}$ and $\mathbf{V}_{\mathrm{psh}}$ whenever convenient.

6 Models of EC and EEC

Our basic effect calculus EC is closely related to Levy's CBPV with stacks. Accordingly, the natural models are given by Levy's adjunction models [22]. While these are most simply presented as locally-indexed categories, see *op. cit.*, we instead use a definition, also discussed in *op. cit.*, based on enriched category theory, since this connects more easily with the models of the enriched effect calculus introduced below. This definition requires a category $\mathbf{V}$ of value types, together with a category $\mathbf{C}$ of computation types enriched over the presheaf category $\widehat{\mathbf{V}}$. We first give the formal definition, and then follow with an intuitive explanation of the structure.

Definition 6.1. An EC *model* comprises: a category with finite products, $\mathbf{V}$; a $\widehat{\mathbf{V}}$-category $\mathbf{C}$ with finite $\widehat{\mathbf{V}}$-products and $\mathbf{y}A$-fold powers for every $\mathbf{V}$-object A; and, a $\widehat{\mathbf{V}}$-adjunction $F \dashv U \colon \mathbf{C} \longrightarrow \mathbf{V}$.

While the definition of EC model requires all the requested structure to be specified; for convenience, we shall normally refer to such a model as simply $F \dashv U \colon \mathbf{C} \longrightarrow \mathbf{V}$. The reason for singling out the enriched adjunction for specific mention, is that it is the one parameter that can be varied in the choice of EC model over a given $\mathbf{V}$ and $\mathbf{C}$. All the remainder of the structure is determined up to isomorphism.

The intuition behind the structure of an EC model is as follows. The category $\mathbf{V}$ models value types, and $\mathbf{C}$ models linear maps between computation types. The reason for requiring $\mathbf{C}$ to be enriched over $\widehat{\mathbf{V}}$ is to model judgements $\Gamma \mid z \colon \underline{\mathsf{A}} \vdash t \colon \underline{\mathsf{B}}$ in non-empty contexts Γ, as elements of the set $\mathbf{C}(\underline{\mathsf{A}}, \underline{\mathsf{B}})(\Gamma)$ or equivalently (by the Yoneda lemma) as morphisms

$$\mathbf{y}(\Gamma) \longrightarrow \mathbf{C}(\underline{\mathsf{A}}, \underline{\mathsf{B}})$$

in $\widehat{\mathbf{V}}$ *cf.* [22]. The $\mathbf{y}A$-fold power of an object $\underline{B}$ of $\mathbf{C}$ models the computation type $A \to \underline{B}$. The left adjoint F maps a value type A to the computation type $!A$. The right adjoint U interprets the coercion from computation types to value types. The other structure (finite products) is self explanatory.

A minor inconvenience with the definition of EC model, as we have given it, is that one needs to assume that the category $\mathbf{V}$ is small, in order to work freely with the presheaf category $\widehat{\mathbf{V}}$. As is standard, to escape this limitation, one can, when necessary, move to a larger universe of sets in order to accommodate large categories $\mathbf{V}$ as small.

Note that all of the structure required in the definition of EC model is assumed to be $\widehat{\mathbf{V}}$-enriched. Thus the definition implicitly refers to $\mathbf{V}$ as the $\widehat{\mathbf{V}}$-category $\mathbf{V}_{\mathrm{psh}}$, as defined in Section 5. Since U is an enriched right adjoint, it preserves all $\widehat{\mathbf{V}}$-enriched limits which exist in $\mathbf{C}$, including powers; hence the following lemma holds.

Lemma 6.2. *Let $F \dashv U \colon \mathbf{C} \longrightarrow \mathbf{V}$ be an* EC *model. Then $\mathbf{V}$ has powers of the form $(U\underline{C})^{\mathbf{y}A}$, with these being given as $U(\underline{C}^{\mathbf{y}A})$. Equivalently, it has internal homs of the form $[A \to U\underline{C}]$.*

Note, however, that we do not assume that $\mathbf{V}$ itself have arbitrary $\mathbf{y}A$-fold powers. That hypothesis would entail $\mathbf{V}$ being cartesian closed, an assumption we do not make, since the value types of EC do not have arbitrary function types.

Definition 6.3. An EC+ *model* is an EC model in which $\mathbf{V}$ has finite $\widehat{\mathbf{V}}$-coproducts, and for which every hom-presheaf $\mathbf{C}(\underline{A}, \underline{B}) \colon \mathbf{V}^{\mathrm{op}} \to \mathbf{Set}$ preserves finite products (that is, finite coproduct diagrams in $\mathbf{V}$ are mapped to finite product diagrams in $\mathbf{Set}$).

In this definition, the requirement that $\mathbf{V}$ (by which, strictly, we mean $\mathbf{V}_{\mathrm{psh}}$) have finite $\widehat{\mathbf{V}}$-coproducts is equivalent to the assertion that the underlying category $\mathbf{V}$ have (ordinary) finite coproducts, and that they are distributive (with respect to $\times$). The distributivity property is, in turn, equivalent to requiring that every hom-presheaf $\mathbf{V}(A, B) \colon \mathbf{V}^{\mathrm{op}} \to \mathbf{Set}$ in $\mathbf{V}$ preserves finite products. The second part of Definition 6.3 requires that the same property hold for hom-presheaves in $\mathbf{C}$. Thus, a final reformulation of the definition is to require that the underlying category $\mathbf{V}$ have finite coproducts and that both $\mathbf{V}$ and $\mathbf{C}$ are enriched in the category of finite-product-preserving (henceforth *fpp*) presheaves. This implies, in particular, that the Yoneda functor $\mathbf{y}$ maps objects of $\mathbf{V}$ to fpp presheaves. Furthermore, when considered as a functor into the category of fpp presheaves, $\mathbf{y}$ preserves finite coproducts. It is this latter fact, which does not hold when $\mathbf{y}$ is considered as a functor into arbitrary presheaves, which necesitates the restriction of hom-objects to fpp presheaves in Definition 6.3.

In the sequel, we shall use the fpp-presheaf structure in the following form. For an fpp-presheaf H in $\mathbf{V}$, there is a unique presheaf map

$$\hat{?} \colon \mathbf{y}0 \longrightarrow H \ , \tag{25}$$

and there is a one-to-one correspondence between pairs of presheaf maps

$$f\colon \mathbf{y}A \longrightarrow H \qquad g\colon \mathbf{y}B \longrightarrow H$$

and maps

$$\hat{[}f, g\hat{]}\colon \mathbf{y}(A+B) \longrightarrow H \ . \tag{26}$$

(These facts are direct consequences of the one-to-one correpondence between maps $\mathbf{y}A \longrightarrow H$ and elements of $H(A)$ asserted by the Yoneda lemma.)

As mentioned above, EC and EC+ models provide enriched-category formulations of Levy's *adjunction models* for CBPV [22]. We refer to [22] for further discussion of such structure and of alternative ways to formulate it. Following [22] we can show a close relation between EC models and Moggi's monad-based metalanguage models [30]. As is standard, we say that a category $\mathbf{V}$ with finite products, carrying a strong monad T, has *Kleisli exponentials* if it has internal homs of the form $[A \to TB]$.

Proposition 6.4 (cf. Examples 4.9 and 5.7 of [22])**.** *Let $F \dashv U\colon \mathbf{C} \longrightarrow \mathbf{V}$ be an* EC *model. The composite UF carries the structure of a strong monad on $\mathbf{V}$, with respect to which $\mathbf{V}$ has Kleisli exponentials.*

Conversely, let T be a strong monad on a category $\mathbf{V}$ with finite products such that $\mathbf{V}$ has Kleisli exponentials. Let $\mathbf{C}$ be the full subcategory of the Eilenberg-Moore category $\mathbf{V}^T$ on finite products of powers of free algebras (powers of free algebras exist by the assumption of Kleisli exponentials). Then the adjunction $F \dashv U\colon \mathbf{C} \longrightarrow \mathbf{V}$ (obtained by cutting down the canonical adjunction $F \dashv U\colon \mathbf{V}^T \longrightarrow \mathbf{V}$) enriches to an EC *model.*

Proof. By the general theory of enriched categories, the enriched adjunction induces a $\widehat{\mathbf{V}}$-enriched monad structure on $\mathbf{V}$. Applying the techniques of [18] to the current setting, and using elementary properties of the Yoneda functor, one sees that such enrichments are equivalent to strengths. More explicitly, the components of the strength can be described as the result of applying the map

$$\mathbf{V}(A\times B, A\times B) = \mathbf{V}_{\mathrm{psh}}(A, A\times B)(B) \xrightarrow{T_{A,A\times B}(B)} \mathbf{V}_{\mathrm{psh}}(TA, T(A\times B))(B) = \mathbf{V}(TA\times B, T(A\times B))$$

to $id_{A\times B}$.

The second statement of the proposition is [22, Examples 4.9 and 5.7]. □

Remark 6.5. In the case that $\mathbf{V}$ is cartesian closed, the canonical adjunction $F \dashv U\colon \mathbf{V}^T \longrightarrow \mathbf{V}$ itself enriches to an EC model. This also happens if $\mathbf{V}$ has Kleisli exponentials and all idempotents in $\mathbf{V}$ split.

Next, we turn to models of the *enriched* effect calculus, EEC. In this calculus, the presence of the linear function space $\underline{\mathsf{A}} \multimap \underline{\mathsf{B}}$ as a value type means that the hom-set $\mathbf{C}(\underline{A}, \underline{B})$ needs to live as an object of the category $\mathbf{V}$ of value types itself. Similarly, the presence of arbitrary function types $\mathsf{A} \to \mathsf{B}$ makes it necessary for $\mathbf{V}$ to be closed. Thus it is natural to require all the structure to be enriched over $\mathbf{V}$ itself, rather than $\widehat{\mathbf{V}}$. This helps to make the definition of an EEC model simpler and more natural than the notion of EC model.

Definition 6.6. An EEC *model* comprises: a cartesian closed category $\mathbf{V}$; a $\mathbf{V}$-enriched category $\mathbf{C}$ with powers and copowers, finite products and coproducts; and, a $\mathbf{V}$-adjunction $F \dashv U\colon \mathbf{C} \longrightarrow \mathbf{V}$. An EEC+ model is an EEC model in which $\mathbf{V}$ has finite coproducts.

As in the case of EC models, we shall typically refer to an EEC model as $F \dashv U\colon \mathbf{C} \longrightarrow \mathbf{V}$, since all other structure, though strictly part of the specification of the model, is determined up to isomorphism.

In the above definition, all the structure is intended to be read as being $\mathbf{V}$-enriched. Thus $\mathbf{C}$ has finite $\mathbf{V}$-products and $\mathbf{V}$-coproducts, and, in the case of an EEC+ model, $\mathbf{V}$ has finite $\mathbf{V}$-coproducts. However, in all three cases, it suffices to merely assume ordinary (co)products on the underlying category and $\mathbf{V}$-enrichment follows. For the finite products and coproducts of $\mathbf{C}$, this is a consequence of Lemma 5.6. For the case of coproducts on $\mathbf{V}$, enrichment holds because $\mathbf{V}$ is cartesian closed, so coproducts are distributive, hence $\mathbf{V}$-enriched.

Every EEC model can be reconstrued as an EC model. Since the Yoneda embedding $\mathbf{y}\colon \mathbf{V} \longrightarrow \widehat{\mathbf{V}}$ preserves products, it defines a monoidal functor $(\mathbf{V}, \times, 1) \longrightarrow (\widehat{\mathbf{V}}, \times, 1)$. Hence, we can apply Lemma 5.1 to transport the $\mathbf{V}$-enriched adjunction $F \dashv U\colon \mathbf{C} \longrightarrow \mathbf{V}$ to a $\widehat{\mathbf{V}}$-enriched one. Since $\mathbf{y}$ also preserves exponentials, it holds that ${}_{\mathbf{y}}\mathbf{V}_{\mathrm{self}} \cong \mathbf{V}_{\mathrm{psh}}$. The $\mathbf{V}$-powers of $\mathbf{C}$ define the required powers of ${}_{\mathbf{y}}\mathbf{C}$. In the case that one starts with an EEC+ model, $\mathbf{V}$ is distributive, hence the Yoneda embedding $\mathbf{y}$ defines a product-preserving (so monoidal) functor from $\mathbf{V}$ to fpp presheaves, allowing the $\mathbf{V}$-enriched adjunction to be transported to one enriched in fpp presheaves; cf. the discussion following Definition 6.3. This way, any EEC+ model is reconstrued as an EC+ model.

Unsurprisingly, the converse statement that EC (respectively EC+) models can be construed as EEC (respectively EEC+) models does not hold, since the latter models have structure that the former need not possess. Nevertheless, every EC (respectively EC+) model can be fully embedded in an EEC (respectively EEC+) model. This important fact will be proved in the companion paper on the model theory of EEC [8]. We end this section by describing some of the many naturally occurring examples of EEC models.

A rich source of EEC models is provided by models of ILL. Amongst the various formulations of such models, the most natural for our purposes is that of *linear/nonlinear* model [2], which consists of a cartesian closed category $\mathbf{V}$ (the *intuitionistic category*), a symmetric monoidal closed category $\mathbf{C}$ (the *linear category*), and a symmetric monoidal adjunction $F \dashv G\colon (\mathbf{C}, \otimes, I) \longrightarrow (\mathbf{V}, \times, 1)$. We say that a linear/nonlinear model *has additives* if its linear category has finite products and coproducts.

Proposition 6.7. *Every linear/nonlinear model with additives determines an* EEC *model.*

Proof. Since $\mathbf{C}$ and $\mathbf{V}$ are closed monoidal categories and G is a monoidal functor, Lemma 5.3 is applicable—that is, G canonically enriches to a $\mathbf{V}$-functor $G_{\mathrm{can}}\colon {}_{G}\mathbf{C} \longrightarrow \mathbf{V}$. Similarly, F enriches to a $\mathbf{V}$-functor $F_{\mathrm{enr}}\colon \mathbf{V} \longrightarrow {}_{G}\mathbf{C}$.[7] That there are $\mathbf{V}$-isomorphisms ${}_{G}\mathbf{C}(FA, \underline{C}) \cong \mathbf{V}(A, G\underline{C})$ manifesting a $\mathbf{V}$-adjunction $F_{\mathrm{enr}} \dashv G_{\mathrm{can}}\colon {}_{G}\mathbf{C} \longrightarrow \mathbf{V}$, follows from the strongness of F as a monoidal functor $(\mathbf{V}, \times, 1) \longrightarrow (\mathbf{C}, \otimes, I)$.

It remains to show that ${}_{G}\mathbf{C}$ has powers, copowers and (enriched) finite products and coproducts. The existence of powers and copowers follows from Lemma 5.4. (To reiterate in conventional notation, $A \cdot \underline{C} = FA \otimes \underline{C}$ and $\underline{C}^{A} = FA \multimap \underline{C}$.) Given that all powers and copowers exist, Lemma 5.6 applies: the existence of finite $\mathbf{V}$-products and -coproducts in ${}_{G}\mathbf{C}$ reduces to the existence of finite products and coproducts in the underlying category of ${}_{G}\mathbf{C}$; and, as it happens, ${}_{UG}\mathbf{C} = {}_{U}({}_{G}\mathbf{C})$ is isomorphic to $\mathbf{C}$. (The last assertion is not entirely trivial, as it requires a further invocation of the strongness of F.) □

For any EEC model the composite UF is a $\mathbf{V}$-enriched monad on $\mathbf{V}$, which means exactly that it is strong [18], see also [30, Remark 3.3]. The next two propositions investigate situations

[7] The monoidal adjunction $F \dashv G$ induces a 2-adjunction between ${}_{F}(\cdot)$ and ${}_{G}(\cdot)$, and F_{enr} is simply the $\mathbf{V}$-functor corresponding across this 2-adjunction to the $\mathbf{C}$-functor $F_{\mathrm{can}}\colon {}_{F}\mathbf{V} \longrightarrow \mathbf{C}$.

in which, conversely, strong monads give rise to EEC models via the Eilenberg-Moore adjunction. When $\mathbf{V}$ is the category $\mathbf{Set}$ this happens automatically (and all monads are strong).

Proposition 6.8. *For any monad T on $\mathbf{Set}$, the Eilenberg-Moore adjunction*

$$F^T \dashv U^T \colon \mathbf{Set}^T \longrightarrow \mathbf{Set}$$

is an EEC+ *model.*

Proof. The adjunction trivially enriches, and we just show that $\mathbf{Set}^T$ has the required structure. Products and powers are defined by the usual pointwise constructions. For example, if (Y, θ) is an algebra and X is a set, then the power is the exponent Y^X equipped with the algebra structure defined as the transpose of the composite

$$X \times T(Y^X) \xrightarrow{\mathrm{st}} T(X \times Y^X) \xrightarrow{T(ev)} T(Y) \xrightarrow{\theta} Y$$

where st is the strength of T (recalling that any monad on $\mathbf{Set}$ is strong).

Since $\mathbf{Set}^T$ is cocomplete [24], it has coproducts. Copowers $X \cdot (Y, \theta)$ are X-fold coproducts of (Y, θ) by itself. □

To generalise the previous result to categories different from $\mathbf{Set}$, it is necessary to make sufficient cocompleteness assumptions about the Eilenberg-Moore category.[8]

Proposition 6.9. *If $\mathbf{V}$ is cartesian closed with finite coproducts and finite limits, and T is a strong monad such that the Eilenberg-Moore category $\mathbf{V}^T$ has reflexive coequalisers, then $F^T \dashv U^T \colon \mathbf{V}^T \longrightarrow \mathbf{V}$ is an* EEC+ *model.*

Details of the proof, which is based on ideas from [24], are deferred to [8].

Proposition 6.9 should be compared with [3, Proposition 2.5], which shows that, under similar assumptions, the Eilenberg-Moore adjunction of a commutative monad T forms a linear/nonlinear model. Thus, for commutative monads, Proposition 6.9 follows from a combination of [3, Proposition 2.5] and Proposition 6.7. Importantly, however, Proposition 6.9 applies also to non-commutative monads, such as those needed to model computational effects that are sensitive to their order of invocation.

In the general case of a non-commutative monad, while Proposition 6.9 produces an EEC+ model, it is not the case that the Eilenberg-Moore adjunction is a linear/nonlinear model. Such examples thus illustrate the added generality of EEC+, beyond intuitionistic linear logic. They also serve to explain some of the design decisions of EEC+. For example, even in the case in which $\mathbf{V}$ is $\mathbf{Set}$, for a non-commutative monad T, the set of homomorphisms between two Eilenberg-Moore algebras need not itself carry an algebra structure. In such models, it is therefore not possible to make sense of $\underline{\mathsf{A}} \multimap \underline{\mathsf{B}}$ as a computation type, since such a type would have to be interpreted as an algebra on the set of homomorphisms. However, the type makes perfect sense as a value type, for which the set of homomorphisms itself provides the interpretation.

We end the section with further examples of families of EEC models, in which the computation category need not be the Eilenberg-Moore category.

Proposition 6.10. *Let $\mathbf{V}$ be cartesian closed, and let $\mathbf{C}$ be $\mathbf{V}$-enriched with powers, copowers, finite products and finite coproducts. Let $\underline{S}$ be an object of $\mathbf{C}$. Then the adjunction $(-) \cdot \underline{S} \dashv \mathcal{C}(\underline{S}, -) \colon \mathbf{C} \to \mathbf{V}$ is an* EEC *model. It is an* EEC+ *model if $\mathbf{V}$ has finite coproducts.*

[8] As noted in the proof of Proposition 6.8, every $\mathbf{Set}$-monad T is strong, and its Eilenberg-Moore category $\mathbf{Set}^T$ has all colimits.

In fact, it is a routine exercise of enriched category theory that every EEC model is equivalent to one in the above form with $\underline{S}$ given by $F1$. This semantic analogue of Example 4.2, is further elaborated upon in [8, 28]. The above formulation often allows an easy description of an EEC model. For example, if $\mathbf{C}$ is any complete and cocomplete ordinary (locally small) category then, any choice of object $\underline{S}$ describes an EEC model over **Set**. One case of particular interest, due to its canonical status, is to choose $\mathbf{C}$ to be the free complete and cocomplete category (over the empty set of generators), as determined by Joyal's theory of free bicompletions [16]. We raise the question of whether the equational theory of EEC is complete with respect to the family of semantic interpretations in this free-bicomplete model, obtained by varying the choice of object $\underline{S}$.

The last example is based on the natural adjunction that decomposes the continuations monad $R^{R^{(\cdot)}}$. The observation is that Levy's adjunction model for control effects [21, 22] is automatically a model of full EEC+.

Example 6.11. Let $\mathbf{V}$ be any cartesian-closed category with finite coproducts. Let R be an object of $\mathbf{V}$. The adjunction $R^{(\cdot)} \dashv R^{(\cdot)} \colon \mathbf{V}^{\mathrm{op}} \longrightarrow \mathbf{V}$ determines an EEC+ model. The enrichment is given by defining $\mathbf{V}^{\mathrm{op}}(X, Y)$ to be $[Y \to X]$ in $\mathbf{V}$. The power Y^X in $\mathbf{V}^{\mathrm{op}}$ is the object $X \times Y$, and the copower $X \cdot Y$ is the object $[X \to Y]$.

We illustrate our syntactic choice of including $\underline{\mathsf{A}} \multimap \underline{\mathsf{B}}$ as a value type only, by showing that such continuations models, in general, provide no interpretation of $\underline{\mathsf{A}} \multimap \underline{\mathsf{B}}$ as a computation type. For a counterexample, let $\mathbf{V}$ be **Set**, and let R be the two element set $\mathbf{2}$. (More generally, we write $\mathbf{n}$ for the n-element set $\{0, \ldots, n-1\}$.) For any sets $\underline{\mathsf{A}}$, $\underline{\mathsf{B}}$, considered as computation types, the interpretation of $\underline{\mathsf{A}} \multimap \underline{\mathsf{B}}$ as a value type is the homset $\mathbf{Set}^{\mathrm{op}}(\underline{\mathsf{A}}, \underline{\mathsf{B}}) = \underline{\mathsf{A}}^{\underline{\mathsf{B}}}$. (See Section 7 below for the general interpretation of EEC syntax in a model.) Were $\underline{\mathsf{A}} \multimap \underline{\mathsf{B}}$ to have interpretation as a computation type, its interpretation would have to be a set $\underline{\mathsf{C}}$ for which the right adjoint $R^{(\cdot)} \colon \mathbf{V}^{\mathrm{op}} \longrightarrow \mathbf{V}$ enjoys the property $R^{\underline{\mathsf{C}}} \cong \underline{\mathsf{A}}^{\underline{\mathsf{B}}}$ (because the right adjoint is required to map the interpretation of a computation type to an object isomorphic to its interpretation as a value type, again see Section 7). Now setting $\underline{\mathsf{A}} = \mathbf{3}$ and $\underline{\mathsf{B}} = \mathbf{1}$, the interpretation of $\underline{\mathsf{A}} \multimap \underline{\mathsf{B}}$ as a computation type would have to be a set $\underline{\mathsf{C}}$ for which $\mathbf{2}^{\underline{\mathsf{C}}} \cong \mathbf{3}^{\mathbf{1}} \cong \mathbf{3}$, which is clearly impossible.

7 Soundness and completeness

In this section we show how to interpret our calculi in their models, and we prove the soundness and completeness of the equational theories relative to such interpretations. Due to the use of $\mathbf{V}$-enrichment rather than $\widehat{\mathbf{V}}$-enrichment, the interpretation of the enriched calculi EEC and EEC+ in their models is somewhat more straightforward than that of the basic caluli EC and EC+. So we begin by considering the enriched case.

We thus consider one of the calculi EEC and EEC+. Let $F \dashv U \colon \mathbf{C} \longrightarrow \mathbf{V}$ be a model of the relevant kind (Definition 6.6). In either case, a value type A is interpreted as an object $\mathbf{V}[\![\mathsf{A}]\!]$ of $\mathbf{V}$, and a computation type $\underline{\mathsf{A}}$ is interpreted as a pair $(\mathbf{C}[\![\underline{\mathsf{A}}]\!], s_{\underline{\mathsf{A}}})$ where: $\mathbf{C}[\![\underline{\mathsf{A}}]\!]$ is an object of $\mathbf{C}$, and $s_{\underline{\mathsf{A}}} \colon U(\mathbf{C}[\![\underline{\mathsf{A}}]\!]) \to \mathbf{V}[\![\underline{\mathsf{A}}]\!]$ is an isomorphism in $\mathbf{V}$. The interpretation is determined by specifying objects $\mathbf{V}[\![\alpha]\!] \in \mathbf{V}$ and $\mathbf{C}[\![\underline{\alpha}]\!] \in \mathbf{C}$, which we assume given. The remainder of the interpretation of types is defined in Figure 8. (In the case of EEC, the clauses involving value-type sums should be ignored.) The non-trivial cases in the inductive definition of the isomorphism $s_{\underline{\mathsf{A}}}$ are the cases for unit type, product and function space. These can easily be constructed from the fact that U preserves products and powers, because it is an enriched right adjoint [17].

Terms are interpreted differently depending on whether they are typed with empty stoup or

$$
\begin{array}{rlrl}
\mathbf{V}[\![\underline{\alpha}]\!] &= U(\mathbf{C}[\![\underline{\alpha}]\!]) & & \\
\mathbf{V}[\![1]\!] &= 1 & \mathbf{C}[\![1]\!] &= 1 \\
\mathbf{V}[\![\mathsf{A}\times\mathsf{B}]\!] &= \mathbf{V}[\![\mathsf{A}]\!]\times\mathbf{V}[\![\mathsf{B}]\!] & \mathbf{C}[\![\underline{\mathsf{A}}\times\underline{\mathsf{B}}]\!] &= \mathbf{C}[\![\underline{\mathsf{A}}]\!]\times\mathbf{C}[\![\underline{\mathsf{B}}]\!] \\
\mathbf{V}[\![\mathsf{A}\to\mathsf{B}]\!] &= \mathbf{V}[\![\mathsf{A}]\!]\to\mathbf{V}[\![\mathsf{B}]\!] & \mathbf{C}[\![\mathsf{A}\to\underline{\mathsf{B}}]\!] &= \mathbf{C}[\![\underline{\mathsf{B}}]\!]^{\mathbf{V}[\![\mathsf{A}]\!]} \\
\mathbf{V}[\![!\mathsf{A}]\!] &= U(\mathbf{C}[\![!\mathsf{A}]\!]) & \mathbf{C}[\![!\mathsf{A}]\!] &= F(\mathbf{V}[\![\mathsf{A}]\!]) \\
\mathbf{V}[\![\mathsf{A}+\mathsf{B}]\!] &= \mathbf{V}[\![\mathsf{A}]\!]+\mathbf{V}[\![\mathsf{B}]\!] & & \\
\mathbf{V}[\![0]\!] &= 0 & & \\
\mathbf{V}[\![\underline{\mathsf{A}}\multimap\underline{\mathsf{B}}]\!] &= \mathbf{C}(\mathbf{C}[\![\underline{\mathsf{A}}]\!],\mathbf{C}[\![\underline{\mathsf{B}}]\!]) & & \\
\mathbf{V}[\![!\mathsf{A}\otimes\underline{\mathsf{B}}]\!] &= U(\mathbf{C}[\![!\mathsf{A}\otimes\underline{\mathsf{B}}]\!]) & \mathbf{C}[\![!\mathsf{A}\otimes\underline{\mathsf{B}}]\!] &= \mathbf{V}[\![\mathsf{A}]\!]\cdot\mathbf{C}[\![\underline{\mathsf{B}}]\!] \\
\mathbf{V}[\![\underline{0}]\!] &= U(\mathbf{C}[\![\underline{0}]\!]) & \mathbf{C}[\![\underline{0}]\!] &= 0 \\
\mathbf{V}[\![\underline{\mathsf{A}}\oplus\underline{\mathsf{B}}]\!] &= U(\mathbf{C}[\![\underline{\mathsf{A}}\oplus\underline{\mathsf{B}}]\!]) & \mathbf{C}[\![\underline{\mathsf{A}}\oplus\underline{\mathsf{B}}]\!] &= \mathbf{C}[\![\underline{\mathsf{A}}]\!]+\mathbf{C}[\![\underline{\mathsf{B}}]\!]
\end{array}
$$

Figure 8: Interpretation of EEC+ types.

not. For $\Gamma \mid - \vdash t\colon \mathsf{A}$, the interpretation is a map in $\mathbf{V}$

$$\mathbf{V}[\![t]\!]\colon \mathbf{V}[\![\Gamma]\!] \to \mathbf{V}[\![\mathsf{A}]\!] \tag{27}$$

where $\mathbf{V}[\![\Gamma]\!]$ is the product of the interpretations of the types in Γ. A typing judgement $\Gamma \mid z\colon \underline{\mathsf{A}} \vdash t\colon \underline{\mathsf{B}}$ is interpreted as a morphism:

$$\mathbf{C}[\![t]\!]\colon \mathbf{V}[\![\Gamma]\!] \to \mathbf{C}(\mathbf{C}[\![\underline{\mathsf{A}}]\!],\mathbf{C}[\![\underline{\mathsf{B}}]\!])\,.$$

The interpretation of terms is defined by induction on the typing judgement, and the definition is given in Figure 9. (As before, the clauses involving value-type sums should be ignored in the case of EEC.)

Figure 9 directly follows the structure of Figures 1–3 and 6, from which the types of all subterms can be read off. It also uses the notation, introduced in Section 5, for the (co)pairing and (co)projections within the enriched finite (co)product structures. We also use ! (respectively ?) to denote the unique morphisms into (respectively out of) terminal (respectively initial) objects, and write π_x for the projection from the product $\mathbf{V}[\![\Gamma]\!]$ to $\mathbf{V}[\![\mathsf{A}]\!]$ determined by an entry $x\colon \mathsf{A}$ in Γ. Several clauses refer to the names for morphisms defined in (19)–(23). In addition, we use η to denote the unit of the monad $U \circ F$ on $\mathbf{V}$, and $\Lambda[-]$ and ev to respectively denote Currying and evaluation in the cartesian closed structure of $\mathbf{V}$. For notational simplicity we write $\overline{U}_{\underline{\mathsf{A}},\underline{\mathsf{B}}}$ for the composite below.

$$\mathbf{C}(\mathbf{C}[\![\underline{\mathsf{A}}]\!],\mathbf{C}[\![\underline{\mathsf{B}}]\!]) \xrightarrow{U} U(\mathbf{C}[\![\underline{\mathsf{B}}]\!])^{U(\mathbf{C}[\![\underline{\mathsf{A}}]\!])} \xrightarrow{[s_{\underline{\mathsf{A}}}^{-1} \to s_{\underline{\mathsf{B}}}]} \mathbf{V}[\![\underline{\mathsf{B}}]\!]^{\mathbf{V}[\![\mathsf{A}]\!]}$$

In the interpretation of $!t \otimes u$ we have used

$$\chi_{A,\underline{B}}\colon A \to \mathbf{C}(\underline{B}, A\cdot\underline{B})$$

to denote the uncurried version of $\psi_{A,\underline{B},A\cdot\underline{B}} \circ \ulcorner id_{A\cdot\underline{B}} \urcorner$. Finally, we use

$$d\colon C\times(A+B) \to (C\times A)+(C\times B)$$

to denote the distributivity isomorphism.

$$
\begin{aligned}
\mathbf{V}[\![x]\!] &= \pi_x & \mathbf{C}[\![z]\!] &= \ulcorner id \urcorner \circ\, ! \\
\mathbf{V}[\![*]\!] &= ! & \mathbf{C}[\![*]\!] &= \ulcorner\, ! \urcorner \circ\, ! \\
\mathbf{V}[\![\langle t, u\rangle]\!] &= \langle \mathbf{V}[\![t]\!], \mathbf{V}[\![u]\!]\rangle & \mathbf{C}[\![\langle t, u\rangle]\!] &= \langle -, - \rangle \circ \langle \mathbf{C}[\![t]\!], \mathbf{C}[\![u]\!]\rangle \\
\mathbf{V}[\![\mathsf{fst}(t)]\!] &= \pi_1 \circ \mathbf{V}[\![t]\!] & \mathbf{C}[\![\mathsf{fst}(t)]\!] &= \mathit{comp} \circ \langle \ulcorner \pi_1 \urcorner \circ\, !, \mathbf{C}[\![t]\!]\rangle \\
\mathbf{V}[\![\mathsf{snd}(t)]\!] &= \pi_2 \circ \mathbf{V}[\![t]\!] & \mathbf{C}[\![\mathsf{snd}(t)]\!] &= \mathit{comp} \circ \langle \ulcorner \pi_2 \urcorner \circ\, !, \mathbf{C}[\![t]\!]\rangle \\
\mathbf{V}[\![\lambda x\colon \mathsf{A}.\, t]\!] &= \Lambda[\mathbf{V}[\![t]\!]] & \mathbf{C}[\![\lambda x\colon \mathsf{A}.\, t]\!] &= \xi^{-1} \circ \Lambda[\mathbf{C}[\![t]\!]] \\
\mathbf{V}[\![s(t)]\!] &= \mathit{ev} \circ \langle \mathbf{V}[\![s]\!], \mathbf{V}[\![t]\!]\rangle & \mathbf{C}[\![s(t)]\!] &= \mathit{ev} \circ \langle \xi \circ \mathbf{C}[\![s]\!], \mathbf{V}[\![t]\!]\rangle \\
\mathbf{V}[\![!t]\!] &= \eta \circ \mathbf{V}[\![t]\!]
\end{aligned}
$$

$$
\begin{aligned}
\mathbf{V}[\![\mathsf{let}\ !x\ \mathsf{be}\ t\ \mathsf{in}\ u]\!] &= s_{\underline{\mathsf{B}}} \circ \mathit{ev} \circ \langle U \circ \rho^{-1} \circ \Lambda[s_{\underline{\mathsf{B}}}^{-1} \circ \mathbf{V}[\![u]\!]], \mathbf{V}[\![t]\!]\rangle \\
\mathbf{C}[\![\mathsf{let}\ !x\ \mathsf{be}\ t\ \mathsf{in}\ u]\!] &= \mathit{comp} \circ \langle \rho^{-1} \circ \Lambda[s_{\underline{\mathsf{B}}}^{-1} \circ \mathbf{V}[\![u]\!]], \mathbf{C}[\![t]\!]\rangle
\end{aligned}
$$

$$
\begin{aligned}
\mathbf{V}[\![?_{\mathsf{A}}(t)]\!] &= ? \circ \mathbf{V}[\![t]\!] \\
\mathbf{C}[\![?_{\mathsf{A}}(t)]\!] &= ? \circ \mathbf{V}[\![t]\!] \\
\mathbf{V}[\![\mathsf{inl}_{\mathsf{A},\mathsf{B}}(t)]\!] &= \mathit{inl} \circ \mathbf{V}[\![t]\!] \\
\mathbf{V}[\![\mathsf{inr}_{\mathsf{A},\mathsf{B}}(t)]\!] &= \mathit{inr} \circ \mathbf{V}[\![t]\!] \\
\mathbf{V}[\![\mathsf{case}\, s\, \mathsf{of}\, (\mathsf{inl}(x).\, t; \mathsf{inr}(y).\, u)]\!] &= [\mathbf{V}[\![t]\!], \mathbf{V}[\![u]\!]] \circ d \circ \langle \mathit{id}, \mathbf{V}[\![s]\!]\rangle \\
\mathbf{C}[\![\mathsf{case}\, s\, \mathsf{of}\, (\mathsf{inl}(x).\, t; \mathsf{inr}(y).\, u)]\!] &= [\mathbf{C}[\![t]\!], \mathbf{C}[\![u]\!]] \circ d \circ \langle \mathit{id}, \mathbf{V}[\![s]\!]\rangle
\end{aligned}
$$

$$
\begin{aligned}
\mathbf{V}[\![\lambda^{\circ} x\colon \underline{\mathsf{A}}.\, t]\!] &= \mathbf{C}[\![t]\!] \\
\mathbf{V}[\![s[t]]\!] &= \mathit{ev} \circ \langle \overline{U} \circ \mathbf{V}[\![s]\!], \mathbf{V}[\![t]\!]\rangle & \mathbf{C}[\![s[t]]\!] &= \mathit{comp} \circ \langle \mathbf{V}[\![s]\!], \mathbf{C}[\![t]\!]\rangle \\
\mathbf{V}[\![!t \otimes u]\!] &= \mathit{ev} \circ \langle \overline{U} \circ \chi \circ \mathbf{V}[\![t]\!], \mathbf{V}[\![u]\!]\rangle & \mathbf{C}[\![!t \otimes u]\!] &= \mathit{comp} \circ \langle \chi \circ \mathbf{V}[\![t]\!], \mathbf{C}[\![u]\!]\rangle \\
\mathbf{V}[\![\underline{?}(t)]\!] &= \mathit{ev} \circ \langle \overline{U} \circ \ulcorner ? \urcorner \circ\, !, \mathbf{V}[\![t]\!]\rangle & \mathbf{C}[\![\underline{?}(t)]\!] &= \mathit{comp} \circ \langle \ulcorner ? \urcorner \circ\, !, \mathbf{C}[\![t]\!]\rangle \\
\mathbf{V}[\![\underline{\mathsf{inl}}(t)]\!] &= \mathit{ev} \circ \langle \overline{U} \circ \ulcorner \mathit{inl} \urcorner \circ\, !, \mathbf{V}[\![t]\!]\rangle & \mathbf{C}[\![\underline{\mathsf{inl}}(t)]\!] &= \mathit{comp} \circ \langle \ulcorner \mathit{inl} \urcorner \circ\, !, \mathbf{C}[\![t]\!]\rangle \\
\mathbf{V}[\![\underline{\mathsf{inr}}(t)]\!] &= \mathit{ev} \circ \langle \overline{U} \circ \ulcorner \mathit{inr} \urcorner \circ\, !, \mathbf{V}[\![t]\!]\rangle & \mathbf{C}[\![\underline{\mathsf{inr}}(t)]\!] &= \mathit{comp} \circ \langle \ulcorner \mathit{inr} \urcorner \circ\, !, \mathbf{C}[\![t]\!]\rangle
\end{aligned}
$$

$$
\begin{aligned}
\mathbf{V}[\![\mathsf{let}\ !x \otimes y\ \mathsf{be}\ s\ \mathsf{in}\ t]\!] &= \mathit{ev} \circ \langle \overline{U} \circ \psi^{-1} \circ \Lambda[\mathbf{C}[\![t]\!]], \mathbf{V}[\![s]\!]\rangle \\
\mathbf{C}[\![\mathsf{let}\ !x \otimes y\ \mathsf{be}\ s\ \mathsf{in}\ t]\!] &= \mathit{comp} \circ \langle \psi^{-1} \circ \Lambda[\mathbf{C}[\![t]\!]], \mathbf{C}[\![s]\!]\rangle \\
\mathbf{V}[\![\underline{\mathsf{case}}\, s\, \mathsf{of}\, (\underline{\mathsf{inl}}(x).\, t; \underline{\mathsf{inr}}(y).\, u)]\!] &= \mathit{ev} \circ \langle \overline{U} \circ [-, -] \circ \langle \mathbf{C}[\![t]\!], \mathbf{C}[\![u]\!]\rangle, \mathbf{V}[\![s]\!]\rangle \\
\mathbf{C}[\![\underline{\mathsf{case}}\, s\, \mathsf{of}\, (\underline{\mathsf{inl}}(x).\, t; \underline{\mathsf{inr}}(y).\, u)]\!] &= \mathit{comp} \circ \langle [-, -] \circ \langle \mathbf{C}[\![t]\!], \mathbf{C}[\![u]\!]\rangle, \mathbf{C}[\![s]\!]\rangle
\end{aligned}
$$

$$
\begin{aligned}
\mathbf{V}[\![f(t_1, \ldots, t_k)]\!] &= \mathbf{V}[\![f]\!] \circ \langle \mathbf{V}[\![t_1]\!], \ldots, \mathbf{V}[\![t_k]\!]\rangle \\
\mathbf{V}[\![g(t_1, \ldots, t_k \mid u)]\!] &= \mathit{ev} \circ \langle \overline{U} \circ \mathbf{C}[\![g]\!] \circ \langle \mathbf{V}[\![t_1]\!], \ldots, \mathbf{V}[\![t_k]\!]\rangle, \mathbf{V}[\![u]\!]\rangle \\
\mathbf{C}[\![g(t_1, \ldots, t_k \mid u)]\!] &= \mathit{comp} \circ \langle \mathbf{C}[\![g]\!] \circ \langle \mathbf{V}[\![t_1]\!], \ldots, \mathbf{V}[\![t_k]\!]\rangle, \mathbf{C}[\![u]\!]\rangle
\end{aligned}
$$

Figure 9: Interpretation of EEC+ terms.

The last two equations of Figure 9 incorporate terms from a signature Σ into the interpretation. This requires each operation $f\colon (\mathsf{A}_1,\ldots,\mathsf{A}_k)\to\mathsf{B}$ and $g\colon (\mathsf{A}_1,\ldots,\mathsf{A}_k \mid \underline{\mathsf{B}})\to\underline{\mathsf{C}}$ in Σ to be assigned a specified morphism in $\mathbf{V}$, as below.

$$\mathbf{V}[\![f]\!]\colon \mathbf{V}[\![\mathsf{A}_1]\!]\times\cdots\times\mathbf{V}[\![\mathsf{A}_k]\!]\to\mathbf{V}[\![\mathsf{B}]\!] \tag{28}$$

$$\mathbf{C}[\![g]\!]\colon \mathbf{V}[\![\mathsf{A}_1]\!]\times\cdots\times\mathbf{V}[\![\mathsf{A}_k]\!]\to\mathbf{C}(\mathbf{C}[\![\underline{\mathsf{B}}]\!],\mathbf{C}[\![\underline{\mathsf{C}}]\!]) \tag{29}$$

We now turn to the case of the basic calculi EC and EC+. Accordingly, let $F\dashv U\colon \mathbf{C}\longrightarrow\!\!\!\!\!\rightarrow\mathbf{V}$ be the relevant kind of model (Definitions 6.1 and 6.3). As before, value types A are interpreted as objects $\mathbf{V}[\![\mathsf{A}]\!]$ of $\mathbf{V}$, and computation types $\underline{\mathsf{A}}$ are interpreted as pairs $(\mathbf{C}[\![\underline{\mathsf{A}}]\!], s_{\underline{\mathsf{A}}})$ where: $\mathbf{C}[\![\underline{\mathsf{A}}]\!]$ is an object of $\mathbf{C}$, and $s_{\underline{\mathsf{A}}}\colon U(\mathbf{C}[\![\underline{\mathsf{A}}]\!])\to\mathbf{V}[\![\underline{\mathsf{A}}]\!]$ is an isomorphism in $\mathbf{V}$. Once again, the interpretation is determined by specifying objects $\mathbf{V}[\![\alpha]\!]\in\mathbf{V}$ and $\mathbf{C}[\![\underline{\alpha}]\!]\in\mathbf{C}$. The remaining types are interpreted as in Figure 8, with the exception of function types, $\mathsf{A}\to\underline{\mathsf{B}}$, which we interpret as below. (Of course, the clauses in Figure 8 involving primitives not in the calculus under consideration should be ignored.)

$$\mathbf{V}[\![\mathsf{A}\to\underline{\mathsf{B}}]\!] = \mathbf{V}[\![\mathsf{A}]\!]\to U\mathbf{C}[\![\underline{\mathsf{B}}]\!] \qquad\qquad \mathbf{C}[\![\mathsf{A}\to\underline{\mathsf{B}}]\!] = \mathbf{C}[\![\underline{\mathsf{B}}]\!]^{\mathbf{y}(\mathbf{V}[\![\mathsf{A}]\!])}$$

Here, in the definition of $\mathbf{C}[\![\mathsf{A}\to\underline{\mathsf{B}}]\!]$, the exponent of the power is now $\mathbf{y}(\mathbf{V}[\![\mathsf{A}]\!])$, due to the change of enriching category to $\widehat{\mathbf{V}}$. Also, the function space $[\mathbf{V}[\![\mathsf{A}]\!]\to U\mathbf{C}[\![\underline{\mathsf{B}}]\!]]$ in $\mathbf{V}$, assumed in the definition of $\mathbf{V}[\![\mathsf{A}\to\underline{\mathsf{B}}]\!]$, is given by Lemma 6.2. Using the isomorphism $s_{\underline{\mathsf{B}}}$, this also provides an internal hom $[\mathbf{V}[\![\mathsf{A}]\!]\to\mathbf{V}[\![\underline{\mathsf{B}}]\!]]$, and so agrees with Figure 8, up to isomorphism. Once again, the isomorphisms $s_{\underline{\mathsf{A}}}$ are easily constructed inductively using the preservation of enriched products (this time, the use of Lemma 6.2 in the construction of $\mathbf{V}[\![\mathsf{A}\to\underline{\mathsf{B}}]\!]$ means that the $s_{\mathsf{A}\to\underline{\mathsf{B}}}$ components are identities).

Terms with empty stoup are again interpreted as maps in $\mathbf{V}$, as in (27). Due to the change of enriching category, typing judgements with stoup, $\Gamma \mid z\colon \underline{\mathsf{A}} \vdash t\colon \underline{\mathsf{B}}$, are now interpreted as morphisms in $\widehat{\mathbf{V}}$,

$$\mathbf{C}[\![t]\!]\colon \mathbf{y}(\mathbf{V}[\![\Gamma]\!])\to\mathbf{C}(\mathbf{C}[\![\underline{\mathsf{A}}]\!],\mathbf{C}[\![\underline{\mathsf{B}}]\!])\,.$$

(By the Yoneda lemma, one could equivalently specify $\mathbf{C}[\![t]\!]$ as an element of $\mathbf{C}(\mathbf{C}[\![\underline{\mathsf{A}}]\!],\mathbf{C}[\![\underline{\mathsf{B}}]\!])(\Gamma)$.) The inductive definition of the interpretation of terms is is given in Figure 10. (Again, the clauses involving sums should be ignored in the case of EC.)

Figure 10 makes use of the following additional notation. We write Ξ for the product-preservation isomorphism $\mathbf{y}(\mathbf{V}[\![\Gamma]\!])\times\mathbf{y}(\mathbf{V}[\![\mathsf{A}]\!])\longrightarrow\!\!\!\!\!\rightarrow\mathbf{y}(\mathbf{V}[\![\Gamma]\!]\times\mathbf{V}[\![\mathsf{A}]\!])$ for the Yoneda functor. We use $\mathbf{y}^{-1}$ to denote the inverse image of the Yoneda functor, which exists since the latter is fully faithful. Finally the elimination terms for finite sum types make use of the assumed fpp structure on hom-presheaves in $\mathbf{C}$, in the form of (25) and (26), introduced in the discussion after Definition 6.3.

Operations from a signature Σ are included in Figure 10 by interpreting non-linear operations $f\colon (\mathsf{A}_1,\ldots,\mathsf{A}_k)\to\mathsf{B}$ as before (28), and specifying, for each operation $g\colon (\mathsf{A}_1,\ldots,\mathsf{A}_k \mid \underline{\mathsf{B}})\to\underline{\mathsf{C}}$, a $\widehat{\mathbf{V}}$ morphism:

$$\mathbf{C}[\![g]\!]\colon \mathbf{y}(\mathbf{V}[\![\mathsf{A}_1]\!]\times\cdots\times\mathbf{V}[\![\mathsf{A}_k]\!])\to\mathbf{C}(\mathbf{C}[\![\underline{\mathsf{B}}]\!],\mathbf{C}[\![\underline{\mathsf{C}}]\!])$$

Theorem 7.1 (Soundness and completeness)**.** *Let Σ and $\mathcal{E}$ be a signature and set of equations for one of the calculi* EC, EC+, EEC, EEC+*, henceforth called X. The following statements are equivalent, for two terms $\Gamma\mid\Delta\vdash s,t\colon\mathsf{A}$.*

1. *$\Gamma\mid\Delta\vdash_{\mathcal{E}} s=t\colon\mathsf{A}$ holds in calculus X.*
2. *For every X-model for the signature Σ and equations $\mathcal{E}$, it holds that $\mathbf{V}[\![s]\!]=\mathbf{V}[\![t]\!]$, if Δ is empty, and $\mathbf{C}[\![s]\!]=\mathbf{C}[\![t]\!]$, if Δ is non-empty.*

$$
\begin{aligned}
\mathbf{V}[\![x]\!] &= \pi_x & \mathbf{C}[\![z]\!] &= \ulcorner id \urcorner \circ\, ! \\
\mathbf{V}[\![*]\!] &= ! & \mathbf{C}[\![*]\!] &= \ulcorner ! \urcorner \circ\, ! \\
\mathbf{V}[\![\langle t, u\rangle]\!] &= \langle \mathbf{V}[\![t]\!], \mathbf{V}[\![u]\!]\rangle & \mathbf{C}[\![\langle t, u\rangle]\!] &= \langle -, -\rangle \circ \langle \mathbf{C}[\![t]\!], \mathbf{C}[\![u]\!]\rangle \\
\mathbf{V}[\![\mathsf{fst}(t)]\!] &= \pi_1 \circ \mathbf{V}[\![t]\!] & \mathbf{C}[\![\mathsf{fst}(t)]\!] &= \mathit{comp} \circ \langle \ulcorner \pi_1 \urcorner \circ\, !, \mathbf{C}[\![t]\!]\rangle \\
\mathbf{V}[\![\mathsf{snd}(t)]\!] &= \pi_2 \circ \mathbf{V}[\![t]\!] & \mathbf{C}[\![\mathsf{snd}(t)]\!] &= \mathit{comp} \circ \langle \ulcorner \pi_2 \urcorner \circ\, !, \mathbf{C}[\![t]\!]\rangle \\
\mathbf{V}[\![\lambda x\colon \mathsf{A}.\, t]\!] &= \Lambda[s_{\underline{\mathsf{B}}}^{-1} \circ \mathbf{V}[\![t]\!]] & \mathbf{C}[\![\lambda x\colon \mathsf{A}.\, t]\!] &= \xi^{-1} \circ \Lambda[\mathbf{C}[\![t]\!] \circ \Xi] \\
\mathbf{V}[\![s(t)]\!] &= s_{\underline{\mathsf{B}}} \circ \mathit{ev} \circ \langle \mathbf{V}[\![s]\!], \mathbf{V}[\![t]\!]\rangle & \mathbf{C}[\![s(t)]\!] &= \mathit{ev} \circ \langle \xi \circ \mathbf{C}[\![s]\!], \mathbf{y}\mathbf{V}[\![t]\!]\rangle \\
\mathbf{V}[\![!t]\!] &= \eta \circ \mathbf{V}[\![t]\!]
\end{aligned}
$$

$$
\begin{aligned}
\mathbf{V}[\![\mathsf{let}\ !x\ \mathsf{be}\ t\ \mathsf{in}\ u]\!] &= s_{\underline{\mathsf{B}}} \circ \mathbf{y}^{-1}(\mathit{ev} \circ \langle U \circ \rho^{-1} \circ \Lambda[\mathbf{y}(s_{\underline{\mathsf{B}}}^{-1} \circ \mathbf{V}[\![u]\!]) \circ \Xi], \mathbf{y}\mathbf{V}[\![t]\!]\rangle) \\
\mathbf{C}[\![\mathsf{let}\ !x\ \mathsf{be}\ t\ \mathsf{in}\ u]\!] &= \mathit{comp} \circ \langle \rho^{-1} \circ \Lambda[\mathbf{y}(s_{\underline{\mathsf{B}}}^{-1} \circ \mathbf{V}[\![u]\!]) \circ \Xi], \mathbf{C}[\![t]\!]\rangle
\end{aligned}
$$

$$
\begin{aligned}
\mathbf{V}[\![?_{\mathsf{A}}(t)]\!] &= ? \circ \mathbf{V}[\![t]\!] \\
\mathbf{C}[\![?_{\mathsf{A}}(t)]\!] &= \hat{?} \circ \mathbf{y}\mathbf{V}[\![t]\!] \\
\mathbf{V}[\![\mathsf{inl}_{\mathsf{A},\mathsf{B}}(t)]\!] &= \mathit{inl} \circ \mathbf{V}[\![t]\!] \\
\mathbf{V}[\![\mathsf{inr}_{\mathsf{A},\mathsf{B}}(t)]\!] &= \mathit{inr} \circ \mathbf{V}[\![t]\!] \\
\mathbf{V}[\![\mathsf{case}\, s\, \mathsf{of}\, (\mathsf{inl}(x).\, t; \mathsf{inr}(y).\, u)]\!] &= [\mathbf{V}[\![t]\!], \mathbf{V}[\![u]\!]] \circ d \circ \langle \mathit{id}, \mathbf{V}[\![s]\!]\rangle \\
\mathbf{C}[\![\mathsf{case}\, s\, \mathsf{of}\, (\mathsf{inl}(x).\, t; \mathsf{inr}(y).\, u)]\!] &= \hat{[}\mathbf{C}[\![t]\!], \mathbf{C}[\![u]\!]\hat{]} \circ \mathbf{y}(d \circ \langle \mathit{id}, \mathbf{V}[\![s]\!]\rangle)
\end{aligned}
$$

$$
\begin{aligned}
\mathbf{V}[\![f(t_1, \ldots, t_k)]\!] &= \mathbf{V}[\![f]\!] \circ \langle \mathbf{V}[\![t_1]\!], \ldots, \mathbf{V}[\![t_k]\!]\rangle \\
\mathbf{V}[\![g(t_1, \ldots, t_k \mid u)]\!] &= s_{\underline{\mathsf{C}}} \circ \mathbf{y}^{-1}(\mathit{ev} \circ \langle \mathbf{C}[\![g]\!] \circ \mathbf{y}\langle \mathbf{V}[\![t_1]\!], \ldots, \mathbf{V}[\![t_k]\!]\rangle, \mathbf{y}(s_{\underline{\mathsf{B}}}^{-1} \circ \mathbf{V}[\![u]\!])\rangle) \\
\mathbf{C}[\![g(t_1, \ldots, t_k \mid u)]\!] &= \mathit{comp} \circ \langle \mathbf{C}[\![g]\!] \circ \mathbf{y}\langle \mathbf{V}[\![t_1]\!], \ldots, \mathbf{V}[\![t_k]\!]\rangle, \mathbf{C}[\![u]\!]\rangle
\end{aligned}
$$

Figure 10: Interpretation of EC+ terms.

Soundness ($1 \implies 2$) is proved by the usual induction on derivations of $\Gamma \mid \Delta \vdash_{\mathcal{E}} s = t\colon \mathsf{A}$. We omit the routine argument entirely. As is standard, the completeness implication ($2 \implies 1$) is proved via the construction of syntactic models, whose construction we now outline.

We start with the case that the calculus X is EC, which is slightly more involved due to enrichment over a presheaf category. The syntactic category $\mathbf{V}_{\mathrm{Syn}}$ has as objects value types and as morphisms from A to B terms of the form $x\colon \mathsf{A} \mid - \vdash t\colon \mathsf{B}$ identified up to the equality theory of EC over the equations $\mathcal{E}$. Composition is by substitution. The $\widehat{\mathbf{V}_{\mathrm{Syn}}}$-enriched structure of $\mathbf{V}_{\mathrm{Syn}}$ is given via (24):

$$
\begin{aligned}
\mathbf{V}_{\mathrm{Syn}_{\mathrm{psh}}}(\mathsf{A}, \mathsf{B})(\mathsf{C}) &\cong \mathbf{V}_{\mathrm{Syn}}(\mathsf{A} \times \mathsf{C}, \mathsf{B}) \\
&= \{t \mid\ x\colon \mathsf{A} \times \mathsf{C} \mid - \vdash t\colon \mathsf{B}\}/\sim \\
&\cong \{t \mid\ x\colon \mathsf{A}, y\colon \mathsf{C} \mid - \vdash t\colon \mathsf{B}\}/\sim\ ,
\end{aligned}
$$

where $\sim$ is provable equality over $\mathcal{E}$. For convenience, we henceforth elide the "psh" subscript

and take the enrichment as being defined by:

$$\mathbf{V}_{\mathrm{Syn}}(\mathsf{A},\mathsf{B})(\mathsf{C}) \;=\; \{t \mid\; x\colon \mathsf{A}, y\colon \mathsf{C} \mid\!- \vdash t\colon \mathsf{B}\}/\sim \quad .$$

The $\widehat{\mathbf{V}_{\mathrm{Syn}}}$-enriched category $\mathbf{C}_{\mathrm{Syn}}$ has computation types as objects and as object of morphisms the presheaf $\mathbf{C}_{\mathrm{Syn}}(\underline{\mathsf{A}},\underline{\mathsf{B}})$ where

$$\mathbf{C}_{\mathrm{Syn}}(\underline{\mathsf{A}},\underline{\mathsf{B}})(\mathsf{C}) \;=\; \{t \mid\; y\colon \mathsf{C} \mid z\colon \underline{\mathsf{A}} \vdash t\colon \underline{\mathsf{B}}\}/\sim \quad .$$

Products in $\mathbf{V}_{\mathrm{Syn}}$ and $\mathbf{C}_{\mathrm{Syn}}$ are simply product types and the $\mathbf{y}(\mathsf{A})$ power of $\underline{\mathsf{B}}$ is the computation type $\mathsf{A} \to \underline{\mathsf{B}}$. The right adjoint U_{Syn} is the inclusion of computation types in value types. Its action on hom-presheaves maps an equivalence class $[t] \in \mathbf{C}_{\mathrm{Syn}}(\underline{\mathsf{A}},\underline{\mathsf{B}})(\mathsf{C})$ (given by $y\colon \mathsf{C} \mid z\colon \underline{\mathsf{A}} \vdash t\colon \underline{\mathsf{B}}$) to $[t[x/z]] \in \mathbf{V}_{\mathrm{Syn}}(\underline{\mathsf{A}},\underline{\mathsf{B}})(\mathsf{C})$ (given by $x\colon \underline{\mathsf{A}}, y\colon \mathsf{C} \mid\!- \vdash t[x/z]\colon \underline{\mathsf{B}}$), which works because the "shift" property of terms (Proposition 2.4) preserves equalities. Conversely, the left adjoint F_{Syn} maps a value type A to $!\mathsf{A}$. That this forms an enriched adjunction follows from observing that the maps

$$[t] \mapsto [\mathsf{let}\ !x\ \mathsf{be}\ z\ \mathsf{in}\ t]\colon \mathbf{V}_{\mathrm{Syn}}(\mathsf{A},\underline{\mathsf{B}})(\mathsf{C}) \to \mathbf{C}_{\mathrm{Syn}}(!\mathsf{A},\underline{\mathsf{B}})(\mathsf{C})$$

are isomorphisms. In the case of EC+, one notes further that sum of value types defines a distributive coproduct in $\mathbf{V}_{\mathrm{Syn}}$. Also, the hom-objects for $\mathbf{C}_{\mathrm{Syn}}$ are fpp presheaves because of the isomorphisms

$$\begin{aligned}
\mathbf{C}_{\mathrm{Syn}}(\underline{\mathsf{A}},\underline{\mathsf{B}})(\mathsf{C}_1 + \mathsf{C}_2) \;&=\; \{t \mid\; y\colon \mathsf{C}_1 + \mathsf{C}_2 \mid z\colon \underline{\mathsf{A}} \vdash t\colon \underline{\mathsf{B}}\}/\sim \\
&\cong\; (\{t \mid\; y\colon \mathsf{C}_1 \mid z\colon \underline{\mathsf{A}} \vdash t\colon \underline{\mathsf{B}}\}/\sim) \times (\{t \mid\; y\colon \mathsf{C}_2 \mid z\colon \underline{\mathsf{A}} \vdash t\colon \underline{\mathsf{B}}\}/\sim) \\
&=\; \mathbf{C}_{\mathrm{Syn}}(\underline{\mathsf{A}},\underline{\mathsf{B}})(\mathsf{C}_1) \times \mathbf{C}_{\mathrm{Syn}}(\underline{\mathsf{A}},\underline{\mathsf{B}})(\mathsf{C}_2) \;\;,
\end{aligned}$$

for the example of the binary case. Verifying the correctness of the remainder of the structure is routine.

For the enriched calculi EEC and EEC+, the syntactic model $F_{\mathrm{Syn}} \dashv U_{\mathrm{Syn}}\colon \mathbf{C}_{\mathrm{Syn}} \to \mathbf{V}_{\mathrm{Syn}}$ is constructed in a similar, but not identical way. (Note that we use the same notation $F_{\mathrm{Syn}} \dashv U_{\mathrm{Syn}}\colon \mathbf{C}_{\mathrm{Syn}} \to \mathbf{V}_{\mathrm{Syn}}$ for this model, even though the enrichment is now over $\mathbf{V}_{\mathrm{Syn}}$ rather than $\widehat{\mathbf{V}_{\mathrm{Syn}}}$. The notational ambiguity will always be resolved by the context.) The syntactic category $\mathbf{V}_{\mathrm{Syn}}$ is constructed just as above. However, $\mathbf{V}_{\mathrm{Syn}}$ now carries a cartesian closed structure given by products and function types. The $\mathbf{V}_{\mathrm{Syn}}$-enriched category $\mathbf{C}_{\mathrm{Syn}}$ has as objects all computation types, with the $\mathbf{V}_{\mathrm{Syn}}$-object of morphisms from $\underline{\mathsf{A}}$ to $\underline{\mathsf{B}}$ given by the value type $\underline{\mathsf{A}} \multimap \underline{\mathsf{B}}$. The functors F_{Syn} and U_{Syn} are defined as in the case of EC but in this case the adjunction becomes $\mathbf{V}_{\mathrm{Syn}}$-enriched by the isomorphism (2). Powers and copowers are defined as $\mathsf{A} \to \underline{\mathsf{B}}$ and $!\mathsf{A} \otimes \underline{\mathsf{B}}$ respectively, and the required isomorphisms for these (23) and (22) are both expressed in (3). Finally products and sums of computation types give products and coproducts in $\mathbf{C}_{\mathrm{Syn}}$; and, in the case of EEC+, the coproducts in $\mathbf{V}_{\mathrm{Syn}}$ are given by sums of value types.

To interpret terms, we need to specify interpretations of the basic operations in the signature Σ. Essentially, these are interpreted as themselves. More precisely, in the case of EC (and EC+) a non-linear operation $f\colon (\mathsf{A}_1,\ldots,\mathsf{A}_k) \to \mathsf{B} \in \Sigma$ is interpreted as the equivalence class of

$$x\colon \mathsf{A}_1 \times \cdots \times \mathsf{A}_k \mid\!- \vdash f(\pi_1(x),\ldots,\pi_k(x))\colon \mathsf{B} \;\;,$$

where we have written π_i for the i'th projection out of the product. To interpret a linear operation $g\colon (\mathsf{A}_1,\ldots,\mathsf{A}_k \mid \underline{\mathsf{B}}) \to \underline{\mathsf{C}} \in \Sigma$, we need to pick a morphism

$$\mathbf{y}(\mathsf{A}_1 \times \cdots \times \mathsf{A}_k) \to \mathbf{C}_{\mathrm{Syn}}(\underline{\mathsf{B}},\underline{\mathsf{C}})\,.$$

By the Yoneda lemma, these correspond to elements of $\mathbf{C}_{\mathrm{Syn}}(\underline{\mathsf{B}}, \underline{\mathsf{C}})(\mathsf{A}_1 \times \cdots \times \mathsf{A}_k)$, and we select the equivalence class of $x \colon \mathsf{A}_1 \times \cdots \times \mathsf{A}_k \mid z \colon \underline{\mathsf{B}} \vdash g(\pi_1(x), \ldots, \pi_k(x) \mid z) \colon \underline{\mathsf{C}}$. In the case of EEC, the interpretation of non-linear operations is the same as for EC. Linear operations such as g, with signature as above, are interpreted as the equivalence class of

$$x \colon \mathsf{A}_1 \times \cdots \times \mathsf{A}_k \mid - \vdash \lambda^{\circ} z \colon \underline{\mathsf{B}}.\, g(\pi_1(x), \ldots, \pi_k(x) \mid z) \colon \underline{\mathsf{B}} \multimap \underline{\mathsf{C}} \ .$$

The main lemma needed to prove completeness is that, for any term t, in one of the calculi EC, EC+, EEC, EEC+, the semantic interpretation of t in $F_{\mathrm{Syn}} \dashv U_{\mathrm{Syn}} \colon \mathbf{C}_{\mathrm{Syn}} \to \mathbf{V}_{\mathrm{Syn}}$ is (essentially) given by the equivalence class generated by the term t itself. Because of the different constructions of syntactic models, the formulation for EC and EC+ is slightly different from that for EEC and EEC+. We formally state the lemma in the latter case only, and leave it to the reader to make the adjustment for EC and EC+.

Lemma 7.2. *In the syntactic model $F_{\mathrm{Syn}} \dashv U_{\mathrm{Syn}} \colon \mathbf{C}_{\mathrm{Syn}} \to \mathbf{V}_{\mathrm{Syn}}$ of* EEC *(or* EEC+*), the interpretation of $\mathbf{V}_{\mathrm{Syn}}[\![t]\!]$ of a term $x_1 \colon \mathsf{A}_1, \ldots, x_k \colon \mathsf{A}_k \mid - \vdash t \colon \mathsf{B}$ is the equivalence class of*

$$x \colon \mathsf{A}_1 \times \cdots \times \mathsf{A}_k \mid - \vdash t[\pi_1(x), \ldots, \pi_k(x)/x_1, \ldots, x_k] \colon \mathsf{B} \ ,$$

and the interpretation $\mathbf{C}_{\mathrm{Syn}}[\![u]\!]$ of a term $x_1 \colon \mathsf{A}_1, \ldots, x_k \colon \mathsf{A}_k \mid z \colon \underline{\mathsf{B}} \vdash u \colon \underline{\mathsf{C}}$ is the equivalence class of

$$x \colon \mathsf{A}_1 \times \cdots \times \mathsf{A}_k \mid - \vdash \lambda^{\circ} z \colon \underline{\mathsf{B}}.\, u[\pi_1(x), \ldots, \pi_k(x)/x_1, \ldots, x_k] \colon \underline{\mathsf{B}} \multimap \underline{\mathsf{C}} \ .$$

The lemma is proved by a straightforward induction on the typing derivation for terms. We omit the details. To derive completeness, suppose $\mathbf{V}[\![s]\!] = \mathbf{V}[\![t]\!]$ or $\mathbf{C}[\![s]\!] = \mathbf{C}[\![t]\!]$ (as appropriate) in all models. Then, in particular, this semantic equality holds in the syntactic model $F_{\mathrm{Syn}} \dashv U_{\mathrm{Syn}} \colon \mathbf{C}_{\mathrm{Syn}} \to \mathbf{V}_{\mathrm{Syn}}$. Thus, the terms s, t themselves (modulo the transformation in the above lemma) inhabit the same equivalence class in $\mathbf{V}_{\mathrm{Syn}}$ or $\mathbf{C}_{\mathrm{Syn}}$. That is, they are provably equal over $\mathcal{E}$.

We remark that, in the case of the non-enriched calculi EC and EC+ we could have also proved completeness by showing a correspondence between EC+ models and Levy's adjunction models [22] based on locally-indexed categories, since Appendix A shows that the calculus EC+ is equivalent to Levy's CBPV.

Acknowledgements

We thank Masahito Hasegawa, Matija Pretnar, Sam Staton, the anonymous referees and, especially, Paul Levy for helpful comments. We are also grateful to the special-issue editor, Ian Mackie, for his encouragement and patience.

APPENDICES

A Equivalence of EC+ and Levy's CBPV

In this appendix, we outline the equivalence between the calculus EC+, presented in Section 2, and Levy's *Call-By-Push-Value (CBPV)* [21]. The appendix is not self contained. We describe the main aspects of the equivalence, in terms that a reader who is familiar with Levy's work will understand. In particular, we shall not recall Levy's calculus in detail. Nor shall we give a detailed proof of the equivalence. The verification merely involves lengthy but routine inductions on derivations. Our purpose is rather to formulate the equivalence (whose statement is not

entirely obvious) in enough detail that the interested reader can, if they wish, straightforwardly fill in the details for themselves.

For simplicity, we work with the pure version of EC+, i.e., with empty signature Σ and set of equations $\mathcal{E}$. The equivalence could be extended to include Σ and $\mathcal{E}$ by adding corresponding features to CBPV.

Regarding CBPV, we work with the finitary version from [21],[9] making minor (but obvious) modifications to the syntax (so the notation for CBPV types is closer to our type notation for EC+). As in Section 2, we assume two classes of type constants. We use $\alpha, \beta, \ldots$ to range over *value type constants*, and $\underline{\alpha}, \underline{\beta}, \ldots$ to range over *computation type constants*. We then use $A, B, \ldots$ to range over *value types*, and $\underline{A}, \underline{B}, \ldots$ to range over *computation types*, which are specified by the grammar below:

$$\begin{aligned} A &::= \alpha \mid U\underline{A} \mid 0 \mid A + B \mid 1 \mid A \times B \\ \underline{A} &::= \underline{\alpha} \mid FA \mid \underline{1} \mid \underline{A} \mathbin{\underline{\times}} \underline{B} \mid A \to \underline{B} \end{aligned}$$

Note that value and computation types are disjoint.

The version of CBPV that is relevant to us is CBPV with complex values and stacks, as presented in Chapter 3 of [21]. We shall not review the syntax of terms at all. Instead, we merely recall that this version of CBPV comes with three associated forms of typing judgement:

$$\Gamma \vdash^{\mathsf{v}} V : A \qquad \Gamma \vdash^{\mathsf{c}} M : \underline{A} \qquad \Gamma \mid \underline{A} \vdash^{\mathsf{k}} K : \underline{B} \ . \tag{30}$$

The $\vdash^{\mathsf{v}}$ judgement types values V, the $\vdash^{\mathsf{c}}$ judgement types computations M, and the $\vdash^{\mathsf{k}}$ judgement types stacks K. In all three cases, Γ is a context assigning value types to variables.

To translate from CBPV to EC+, we first define type translations. A CBPV value type A translates to an EC+ value type A^{*v}, and a CBPV computation type $\underline{A}$ translates to an EC+ computation type $\underline{A}^{*c}$, as follows.

$$\begin{aligned} \alpha^{*v} &= \alpha & \qquad \underline{\alpha}^{*c} &= \underline{\alpha} \\ (U\underline{A})^{*v} &= \underline{A}^{*c} & \qquad (FA)^{*c} &= !(A^{*v}) \\ 0^{*v} &= 0 & \qquad \underline{1}^{*c} &= 1 \\ (A + B)^{*v} &= A^{*v} + B^{*v} & \qquad (\underline{A} \mathbin{\underline{\times}} \underline{B})^{*c} &= \underline{A}^{*c} \times \underline{B}^{*c} \\ 1^{*v} &= 1 & \qquad (A \to \underline{B})^{*c} &= A^{*v} \to \underline{B}^{*c} \\ (A \times B)^{*v} &= A^{*v} \times B^{*v} \end{aligned}$$

On terms, each of the three CBPV judgement-forms (30) translates into a corresponding judgement in EC+:

$$\Gamma^{*v} \mid\!- \vdash V^{*v} : A^{*v} \qquad \Gamma^{*v} \mid\!- \vdash M^{*c} : \underline{A}^{*c} \qquad \Gamma^{*v} \mid z : \underline{A}^{*c} \vdash K^{*k} : \underline{B}^{*c} \ ,$$

defined by a routine induction on CBPV derivations. (Here, Γ^{*v} just applies $(\cdot)^{*v}$ to every type in Γ.)

For the converse translation, an EC+ value type A is translated to a CBPV value type $\mathsf{A}^{\dagger v}$,

[9] A similar equivalence can be established between Levy's infinitary CBPV and a corresponding infinitary version of EC+.

and an EC+ computation type $\underline{\mathsf{A}}$ is translated to a CBPV computation type $\underline{\mathsf{A}}^{\dagger c}$, as follows.

$$\begin{array}{rcl@{\qquad}rcl}
\alpha^{\dagger v} &=& \alpha & & & \\
\underline{\alpha}^{\dagger v} &=& U(\underline{\alpha}^{\dagger c}) & \underline{\alpha}^{\dagger c} &=& \underline{\alpha} \\
1^{\dagger v} &=& 1 & 1^{\dagger c} &=& \underline{1} \\
(\mathsf{A} \times \mathsf{B})^{\dagger v} &=& \mathsf{A}^{\dagger v} \times \mathsf{B}^{\dagger v} & (\underline{\mathsf{A}} \times \underline{\mathsf{B}})^{\dagger c} &=& \underline{\mathsf{A}}^{\dagger c} \mathbin{\underline{\times}} \underline{\mathsf{B}}^{\dagger c} \\
(\mathsf{A} \to \underline{\mathsf{B}})^{\dagger v} &=& U(\mathsf{A}^{\dagger v} \to \underline{\mathsf{B}}^{\dagger c}) & (\mathsf{A} \to \underline{\mathsf{B}})^{\dagger c} &=& \mathsf{A}^{\dagger v} \to \underline{\mathsf{B}}^{\dagger c} \\
(!\mathsf{A})^{\dagger v} &=& U(F(\mathsf{A}^{\dagger v})) & (!\mathsf{A})^{\dagger c} &=& F(\mathsf{A}^{\dagger v}) \\
0^{\dagger v} &=& 0 & & & \\
(\mathsf{A} + \mathsf{B})^{\dagger v} &=& \mathsf{A}^{\dagger v} + \mathsf{B}^{\dagger v} & & &
\end{array}$$

In addition, for every EC+ computation type $\underline{\mathsf{A}}$, mutually inverse CBPV terms,

$$y : U(\underline{\mathsf{A}}^{\dagger c}) \vdash^{\mathsf{v}} I_{\underline{\mathsf{A}}} : \underline{\mathsf{A}}^{\dagger v} \qquad x : \underline{\mathsf{A}}^{\dagger v} \vdash^{\mathsf{v}} I_{\underline{\mathsf{A}}}^{-1} : U(\underline{\mathsf{A}}^{\dagger c}) \ ,$$

are defined by induction on $\underline{\mathsf{A}}$, witnessing an isomorphism between $\underline{\mathsf{A}}^{\dagger v}$ and $U(\underline{\mathsf{A}}^{\dagger c})$. (The isomorphisms are needed because the types are not, in general, identical. For example, we have $(\underline{\alpha} \times \underline{\beta})^{\dagger v} = U\underline{\alpha} \times U\underline{\beta}$, and $U((\underline{\alpha} \times \underline{\beta})^{\dagger c}) = U(\underline{\alpha} \mathbin{\underline{\times}} \underline{\beta})$.)

The isomorphism $I_{\underline{\mathsf{A}}}$ and its inverse are used in the translation of terms. Specifically, each of the three judgement possibilities for EC+,

$$\Gamma \mid - \vdash s : \mathsf{A} \qquad \Gamma \mid - \vdash t : \underline{\mathsf{A}} \qquad \Gamma \mid z : \underline{\mathsf{A}} \vdash u : \underline{\mathsf{B}}$$

is respectively translated to a CBPV judgement

$$\Gamma^{\dagger v} \vdash^{\mathsf{v}} s^{\dagger v} : \mathsf{A}^{\dagger v} \qquad \Gamma^{\dagger v} \vdash^{\mathsf{c}} t^{\dagger c} : \underline{\mathsf{A}}^{\dagger c} \qquad \Gamma^{\dagger v} \mid \underline{\mathsf{A}}^{\dagger c} \vdash^{\mathsf{k}} u^{\dagger k} : \underline{\mathsf{B}}^{\dagger c} \ .$$

(Note that the second EC+ judgement is a special case of the first, and hence obtains two different translations into CBPV.)

To formulate the equivalence between the two systems, we note first that, for every EC+ value type A, it holds that $\mathsf{A}^{\dagger v *v} = \mathsf{A}$, and, for every EC+ computation type $\underline{\mathsf{A}}$, it holds that $\underline{\mathsf{A}}^{\dagger c *c} = \underline{\mathsf{A}}$. (The two equalities are easily proved simultaneously by induction on types.) In the other direction, for every CBPV value type A and computation type $\underline{A}$, there exist isomorphisms:

$$\begin{array}{ll}
x : A^{*v\,\dagger v} \vdash^{\mathsf{v}} J_A : A & x : A \vdash^{\mathsf{v}} J_A^{-1} : A^{*v\,\dagger v} \\
\mid \underline{A}^{*c\,\dagger c} \vdash^{\mathsf{k}} L_{\underline{A}} : \underline{A} & \mid \underline{A} \vdash^{\mathsf{k}} L_{\underline{A}}^{-1} : \underline{A}^{*c\,\dagger c}
\end{array} \ ,$$

defined using the isomorphisms $I_{\underline{\mathsf{A}}}$ mentioned above. (The critical case that shows the need for the isomorphisms is $(U\underline{A})^{*v\,\dagger v} = \underline{A}^{*c\,\dagger v} \cong U(\underline{A}^{*c\,\dagger c})$, where the isomorphism is given by $I^{-1}_{\underline{A}^{*c}}$.)

Proposition A.1 (Equivalence of EC+ and CBPV)**.** *Given* CBPV *equalities*

$$\Gamma \vdash^{\mathsf{v}} V_1 = V_2 : A \qquad \Gamma \vdash^{\mathsf{c}} M_1 = M_2 : \underline{A} \qquad \Gamma \mid \underline{A} \vdash^{\mathsf{k}} K_1 = K_2 : \underline{B} \ ,$$

there are corresponding EC+ *equalities*

$$\Gamma^{*v} \mid - \vdash V_1^{*v} = V_2^{*v} : A^{*v} \qquad \Gamma^{*v} \mid - \vdash M_1^{*c} = M_2^{*c} : \underline{A}^{*c} \qquad \Gamma^{*v} \mid z : \underline{A}^{*c} \vdash K_1^{*k} = K_2^{*k} : \underline{B}^{*c} \ .$$

Conversely, given EC+ *equalities*

$$\Gamma \mid - \vdash s_1 = s_2 : \mathsf{A} \qquad \Gamma \mid - \vdash t_1 = t_2 : \underline{\mathsf{A}} \qquad \Gamma \mid z : \underline{\mathsf{A}} \vdash u_1 = u_2 : \underline{\mathsf{B}} \ ,$$

there are corresponding CBPV *equalities*

$$\Gamma^{\dagger v} \vdash^{\mathsf{v}} s_1^{\dagger v} = s_2^{\dagger v} \colon \mathsf{A}^{\dagger v} \qquad \Gamma^{\dagger v} \vdash^{\mathsf{c}} t_1^{\dagger c} = t_2^{\dagger c} \colon \underline{\mathsf{A}}^{\dagger c} \qquad \Gamma^{\dagger v} \mid \underline{\mathsf{A}}^{\dagger c} \vdash^{\mathsf{k}} u_1^{\dagger k} = u_2^{\dagger k} \colon \underline{\mathsf{B}}^{\dagger c} \ .$$

Furthermore, for CBPV *terms,*

$$\Gamma \vdash^{\mathsf{v}} V \colon A \qquad \Gamma \vdash^{\mathsf{c}} M \colon \underline{A} \qquad \Gamma \mid \underline{A} \vdash^{\mathsf{k}} K \colon \underline{B} \ ,$$

there are corresponding CBPV *equalities,*

$$\Gamma \vdash^{\mathsf{v}} V = J_A[V^{*v\,\dagger v}/x] \colon A \qquad \Gamma \vdash^{\mathsf{c}} M = M^{*c\,\dagger c} \bullet L_{\underline{B}} \colon \underline{A} \qquad \Gamma \mid \underline{A} \vdash^{\mathsf{k}} K = L_{\underline{A}}^{-1} +\!\!+ K^{*k\,\dagger k} +\!\!+ L_{\underline{B}} \colon \underline{B} \ ,$$

where $\bullet$ *and* $+\!\!+$ *are the dismantling and concatenation operations on* CBPV *stacks, see [21, §2.3.5]. Conversely, given* EC+ *terms*

$$\Gamma \mid - \vdash s \colon \mathsf{A} \qquad \Gamma \mid - \vdash t \colon \underline{\mathsf{A}} \qquad \Gamma \mid z \colon \underline{\mathsf{A}} \vdash u \colon \underline{\mathsf{B}} \ ,$$

there are corresponding EC+ *equalities*

$$\Gamma \mid - \vdash s = s^{\dagger v *v} \colon \mathsf{A} \qquad \Gamma \mid - \vdash t = t^{\dagger c *c} \colon \underline{\mathsf{A}} \qquad \Gamma \mid z \colon \underline{\mathsf{A}} \vdash u = u^{\dagger k *k} \colon \underline{\mathsf{B}} \ .$$

We end this appendix with some remarks on the equivalence established by Proposition A.1. We believe that the very fact that CBPV with complex stacks is equivalent to a calculus with a presentation as compact as that of EC+ is interesting, given the weighty nature of its original formulation in [21]. One way of interpreting the equivalence is that the additional notational bureaucracy of CBPV (the type constructor U, the associated "thunk" and "force" operations, the syntactic distinction between computation-type and value-type products, the different notation for terms and stacks) can be automatically reconstructed from the streamlined version offered by EC+, up to isomorphism.

In spite of its conciseness, our approach to formulating the calculi in this paper is not a panacea. For some purposes, it is helpful to restore some of the syntactic distinctions of CBPV into effect calculi. For example, results in both [9, 8] are simplified by using a formulation of EEC in which value-type products and function spaces are distinguished from computation-type products and function spaces. We mention briefly why this is the case in [8]. There, two kinds of morphism of model are considered: morphisms that preserve structure strictly (on the nose); and morphisms that preserve structure up to isomorphism. The latter are mathematically more natural, and the syntactic model we constructed in the completeness proof of Section 7 enjoys an appropriate (2-categorical) initiality property with respect to such morphisms, which characterises it up to equivalence. However, in order to study properties of initial models, it proves mathematically convenient to have a model that is initial (in the usual 1-categorical sense) with respect to strict morphisms. Such a model is most easily constructed as a syntactic model for a variant of EEC in which notational distinctions are made between computation-type and value-type products and function spaces. Related to this discussion, we remark that Proposition A.1 above is a syntactic formulation of the semantic statement: there is a non-strict equivalence of models between the strictly-initial EC+ model (which is given by the syntax of CBPV) and the syntactic EC+ model constructed in Section 7.

B Normalization and syntactic conservativity

In this appendix, we provide the main details of the proof of Theorem 4.3, the syntactic conservativity of each larger system amongst EEC+, EEC, EC+ and EC over each smaller one.

$$
\begin{aligned}
\mathsf{fst}(\langle t, u\rangle) &\to t \\
\mathsf{snd}(\langle t, u\rangle) &\to u \\
(\lambda x\colon \mathsf{A}.\, t)u &\to t[u/x] \\
\mathsf{let}\ !x\ \mathsf{be}\ !u\ \mathsf{in}\ t &\to t[u/x] \\
\mathsf{case}\,\mathsf{inl}(s)\,\mathsf{of}\,(\mathsf{inl}(x).\, t; \mathsf{inr}(x).\, t') &\to t[s/x] \\
\mathsf{case}\,\mathsf{inr}(s)\,\mathsf{of}\,(\mathsf{inl}(x).\, t; \mathsf{inr}(x).\, t') &\to t'[s/x] \\
(\lambda^{\circ} x\colon \underline{\mathsf{A}}.\, t)[u] &\to t[u/x] \\
\mathsf{let}\ !x \otimes y\ \mathsf{be}\ !s \otimes u\ \mathsf{in}\ t &\to t[s,u/x,y] \\
\underline{\mathsf{case}}\,\underline{\mathsf{inl}}(s)\,\mathsf{of}\,(\underline{\mathsf{inl}}(x).\, t; \underline{\mathsf{inr}}(x).\, t') &\to t[s/x] \\
\underline{\mathsf{case}}\,\underline{\mathsf{inr}}(s)\,\mathsf{of}\,(\underline{\mathsf{inl}}(x).\, t; \underline{\mathsf{inr}}(x).\, t') &\to t'[s/x]
\end{aligned}
$$

$$
\begin{aligned}
E[\mathsf{let}\ !x\ \mathsf{be}\ u\ \mathsf{in}\ t] &\to \mathsf{let}\ !x\ \mathsf{be}\ u\ \mathsf{in}\ E[t] \\
E[?(t)] &\to ?(t) \\
E[\mathsf{case}\, s\,\mathsf{of}\,(\mathsf{inl}(x).\, t; \mathsf{inr}(x).\, t')] &\to \mathsf{case}\, s\,\mathsf{of}\,(\mathsf{inl}(x).\, E[t]; \mathsf{inr}(x).\, E[t']) \\
E[\mathsf{let}\ !x \otimes y\ \mathsf{be}\ u\ \mathsf{in}\ t] &\to \mathsf{let}\ !x \otimes y\ \mathsf{be}\ u\ \mathsf{in}\ E[t] \\
E[\underline{?}(t)] &\to \underline{?}(t) \\
E[\underline{\mathsf{case}}\, s\,\mathsf{of}\,(\underline{\mathsf{inl}}(x).\, t; \underline{\mathsf{inr}}(x).\, t')] &\to \underline{\mathsf{case}}\, s\,\mathsf{of}\,(\underline{\mathsf{inl}}(x).\, E[t]; \underline{\mathsf{inr}}(x).\, E[t'])
\end{aligned}
$$

$$
\frac{t \to u}{C[t] \to C[u]}
$$

Figure 11: Rewrite relation on terms of EEC+. Here $E[-]$ range over elimination frames, and $C[-]$ ranges over all one-hole contexts.

This is achieved by providing a normalizing rewrite relation between terms of full EEC+, which restricts to a rewrite relation on EEC, EC and EC+. Since every term is equal to its normal form, syntactic conservativity can be obtained by establishing conservativity for normal forms, which follows directly from the structure of normal forms (essentially, because the typing derivations for normal-form terms enjoy a "subtype property" analogous to the *subformula property* in proof theory), see Lemma B.7 below.

The rewrite relation is defined in Figure 11. The figure uses the concept of *elimination frame* $E[-]$ which is defined by the grammar

$$
\begin{aligned}
E[-] ::= \mathsf{fst}([-]) \mid \mathsf{snd}([-]) \mid [-]\,(t) \mid \mathsf{let}\ !x\ \mathsf{be}\ [-]\ \mathsf{in}\ t \mid ?([-]) \mid \mathsf{case}\,[-]\,\mathsf{of}\,(\mathsf{inl}(x).\, t; \mathsf{inr}(x).\, t') \mid \\
[-][t] \mid \mathsf{let}\ !x \otimes y\ \mathsf{be}\ [-]\ \mathsf{in}\ t \mid \underline{?}([-]) \mid \underline{\mathsf{case}}\,[-]\,\mathsf{of}\,(\underline{\mathsf{inl}}(x).\, t; \underline{\mathsf{inr}}(x).\, t')\ .
\end{aligned}
$$

A term t is a *redex* if it matches the left-hand side of one of the 16 axioms of Figure 11. The first ten axioms are *beta reductions*, and the last six are *permutative reductions*.

Although not explicitly visible in Figure 11, terms in a signature Σ are included within the scope of the rewrite relation. They do appear implicitly in the figure, since meta-variables for terms $(t, u, \dots)$ and one-hole contexts $C[-]$ both range over such terms that may contain operations from Σ. In this appendix, in order to be clear about the role of the signature, we annotate typing and equational judgements with Σ. This is appropriate for equational judgements, since equations will always be provable from an empty set $\mathcal{E}$ of equational axioms.

A basic lemma about the rewrite relation says that it is both type- and equationally sound.

Lemma B.1. *Let X be any of the systems* EEC+, EEC, EC *and* EC+. *If* $\Gamma \mid \Delta \vdash_\Sigma t \colon \mathsf{A}$ *in* X *and* $t \to u$ *then both* $\Gamma \mid \Delta \vdash_\Sigma u \colon \mathsf{A}$ *and* $\Gamma \mid \Delta \vdash_\Sigma t = u \colon \mathsf{A}$ *hold in* X.

We omit the proof, which is a straightforward case analysis.

B.1 Strong normalization

In this subsection, we prove that the rewrite relation is strongly normalizing. For brevity, we do this by reducing strong normalization for EEC+ to strong normalization for the simply-typed λ-calculus with finite products and finite sums, and with beta and permutative reductions, which is a standard result. To avoid the need for introducing yet another calculus formally, we consider the the latter system as the subsystem of EC+ obtained by restricting the types to the *simple types*, which are those given by the grammar:

$$\sigma \ ::= \ \alpha \mid \underline{\alpha} \mid 1 \mid \sigma \times \sigma \mid \sigma \to \sigma \mid 0 \mid \sigma + \sigma \ ,$$

and the terms to those constructed using the typing rules associated with the above type constructors. We write $\lambda+$ for the resulting subsystem of EC+, and we say that $\Gamma \vdash t \colon \sigma$ in $\lambda+$ to mean that the typing judgment $\Gamma \mid - \vdash t \colon \sigma$ is derivable in this subsystem. The rewrite relation $\to$, restricted to terms of $\lambda+$, is exactly the standard rewrite relation of beta and permutative (also called *commuting*) reductions for the simply-typed λ-calculus with sum and product types, as presented, for example, in [12, Chapter 10]. As stated in *op. cit.*, this rewrite relation on $\lambda+$ is strongly normalizing, as can be shown using the techniques of [34, 15] (a full proof for exactly the rewrite relation considered here appears in Appendix A of [35]).

To reduce strong normalization for EEC+ to the simply-typed case, we give a translation from EEC+ into $\lambda+$. The translation maps any EEC+ value type A to a simple type A^*, as specified below.

$$\begin{aligned}
\alpha^* &= \alpha & 0^* &= 0 \\
\underline{\alpha}^* &= \underline{\alpha} & (\mathsf{A} + \mathsf{B})^* &= \mathsf{A}^* + \mathsf{B}^* \\
1^* &= 1 & (\underline{\mathsf{A}} \multimap \underline{\mathsf{B}})^* &= \underline{\mathsf{A}}^* \to \underline{\mathsf{B}}^* \\
(\mathsf{A} \times \mathsf{B})^* &= \mathsf{A}^* \times \mathsf{B}^* & (!\mathsf{A} \otimes \underline{\mathsf{B}})^* &= (\mathsf{A}^* \times \underline{\mathsf{B}}^*) + 0 \\
(\mathsf{A} \to \mathsf{B})^* &= \mathsf{A}^* \to \mathsf{B}^* & \underline{0}^* &= 0 \\
(!\mathsf{A})^* &= \mathsf{A}^* + 0 & (\underline{\mathsf{A}} \oplus \underline{\mathsf{B}})^* &= \underline{\mathsf{A}}^* \oplus \underline{\mathsf{B}}^*
\end{aligned}$$

Computation types are translated as value types.

The translation from an EEC+ term t to a $\lambda+$ term t^* is defined in Figure 12. In this, operations $f \colon (\mathsf{A}_1, \ldots, \mathsf{A}_k) \to \mathsf{B}$ and $g \colon (\mathsf{A}_1, \ldots, \mathsf{A}_k \mid \underline{\mathsf{B}}) \to \underline{\mathsf{C}}$ from a signature Σ are translated relative to an induced set Σ^* of simply-typed constants

$$\begin{aligned}
f^* &\colon \mathsf{A}_1^* \times \cdots \times \mathsf{A}_k^* \to \mathsf{B}^* \\
g^* &\colon \mathsf{A}_1^* \times \cdots \times \mathsf{A}_k^* \times \underline{\mathsf{B}}^* \to \underline{\mathsf{C}}^* \ ,
\end{aligned}$$

using an encoding of k-ary products using binary products and the terminal type 1.

Lemma B.2. $(t[u/x])^* = t^*[u^*/x]$.

Lemma B.3. *If* $\Gamma \mid \Delta \vdash_\Sigma t \colon \mathsf{A}$ *in* EEC+ *then* $\Gamma^*, \Delta^* \vdash_{\Sigma^*} t^* \colon \mathsf{A}^*$ *in* $\lambda+$.

We omit the proofs, which are straightforward inductions on the structure of t.

As is standard, we write $\to^+$ for the transitive closure of $\to$, and $\to^*$ for the reflexive-transitive closure.

$$
\begin{aligned}
x^* &= x \\
^ &= * \\
\langle t, u\rangle^* &= \langle t^*, u^*\rangle \\
(\mathsf{fst}(t))^* &= \mathsf{fst}(t^*) \\
(\mathsf{snd}(t))^* &= \mathsf{snd}(t^*) \\
(\lambda x\colon \mathsf{A}.\, t)^* &= \lambda x\colon \mathsf{A}^*.\, t^* \\
(s(t))^* &= s^*(t^*) \\
(!t)^* &= \mathsf{inl}(t^*) \\
(\mathsf{let}\ !x\ \mathsf{be}\ t\ \mathsf{in}\ u)^* &= \mathsf{case}\, t^*\, \mathsf{of}\, (\mathsf{inl}(x).\, u^*; \mathsf{inr}(y).\, ?(y)) \\
(?(t))^* &= ?(t^*) \\
(\mathsf{inl}(t))^* &= \mathsf{inl}(t^*) \\
(\mathsf{inr}(t))^* &= \mathsf{inr}(t^*) \\
(\mathsf{case}\, s\, \mathsf{of}\, (\mathsf{inl}(x).\, t; \mathsf{inr}(y).\, u))^* &= \mathsf{case}\, s^*\, \mathsf{of}\, (\mathsf{inl}(x).\, t^*; \mathsf{inr}(y).\, u^*) \\
(\lambda^\circ z\colon \underline{\mathsf{A}}.\, t)^* &= \lambda z\colon \underline{\mathsf{A}}^*.\, t^* \\
(s[t])^* &= s^*(t^*) \\
(!t \otimes u)^* &= \mathsf{inl}(\langle t^*, u^*\rangle) \\
(\mathsf{let}\ !x \otimes y\ \mathsf{be}\ s\ \mathsf{in}\ t)^* &= \mathsf{case}\, s^*\, \mathsf{of}\, (\mathsf{inl}(z).\, t^*[\mathsf{fst}(z), \mathsf{snd}(z)/x, y]; \mathsf{inr}(w).\, ?(w)) \\
(\underline{?}(t))^* &= ?(t^*) \\
(\underline{\mathsf{inl}}(t))^* &= \mathsf{inl}(t^*) \\
(\underline{\mathsf{inr}}(t))^* &= \mathsf{inr}(t^*) \\
(\underline{\mathsf{case}}\, s\, \mathsf{of}\, (\underline{\mathsf{inl}}(x).\, t; \underline{\mathsf{inr}}(y).\, u))^* &= \mathsf{case}\, s^*\, \mathsf{of}\, (\mathsf{inl}(x).\, t^*; \mathsf{inr}(y).\, u^*) \\
f(t_1, \ldots, t_k)^* &= f^*\langle t_1^*, \ldots, t_k^*\rangle \\
g(t_1, \ldots, t_k \mid u)^* &= g^*\langle t_1^*, \ldots, t_k^*, u^*\rangle
\end{aligned}
$$

Figure 12: Translation of EEC+ terms to simply-typed terms

Lemma B.4. *If* $\Gamma \mid \Delta \vdash_\Sigma t\colon \mathsf{A}$ *in* EEC+ *and* $t \to u$ *then* $t^* \to^+ u^*$.

Proof. For the ten beta reductions, the property is easily checked on a case-by-case basis. We consider one case.

$$
\begin{aligned}
(\mathsf{let}\ !x \otimes y\ \mathsf{be}\ !s \otimes u\ \mathsf{in}\ t)^* &= \mathsf{case}\, \mathsf{inl}(\langle s^*, u^*\rangle)\, \mathsf{of}\, (\mathsf{inl}(z).\, t^*[\mathsf{fst}(z), \mathsf{snd}(z)/x, y]; \mathsf{inr}(w).\, ?(w)) \\
&\to t^*[\mathsf{fst}(\langle s^*, u^*\rangle), \mathsf{snd}(\langle s^*, u^*\rangle)/x, y] \\
&\to^* t^*[s^*, u^*/x, y] \\
&= (t[s,u/x,y])^* \ ,
\end{aligned}
$$

where the last step is by Lemma B.2.

We similarly consider just one case of a permutative reduction, and show that

$$
(E[\mathsf{let}\ !x \otimes y\ \mathsf{be}\ u\ \mathsf{in}\ t])^* \to^+ (\mathsf{let}\ !x \otimes y\ \mathsf{be}\ u\ \mathsf{in}\ E[t])^* \ . \tag{31}
$$

The strategy, which is also employed for the other permutative reductions, is to split this into two subcases, one for the case in which the elimination frame $E[-]$ is $\mathsf{let}\ !x_0 \otimes y_0\ \mathsf{be}\ [-]\ \mathsf{in}\ t_0$,

and one for all other possible elimination frames. We deal with the latter subcase first. For such elimination frames, $E[-]$, we note that there exists an elimination frame $E^*[-]$ such that $(E[s])^* = E^*[s^*]$, for all terms s. For example,

$$(\mathsf{let}\ !x\ \mathsf{be}\ [-]\ \mathsf{in}\ t)^* \ =\ \mathsf{case}\,[-]\,\mathsf{of}\,(\mathsf{inl}(x).\,t^*;\mathsf{inr}(y).\,?(y))\ .$$

Thus, (31) has a uniform justification:

$$\begin{aligned}
(E[\mathsf{let}\ !x\otimes y\ \mathsf{be}\ u\ \mathsf{in}\ t])^* \ &=\ E^*[\mathsf{case}\,u^*\,\mathsf{of}\,(\mathsf{inl}(z).\,t^*[\mathsf{fst}(z),\mathsf{snd}(z)/x,y];\mathsf{inr}(w).\,?(w))]\\
&\to\ \mathsf{case}\,u^*\,\mathsf{of}\,(\mathsf{inl}(z).\,E^*[t^*[\mathsf{fst}(z),\mathsf{snd}(z)/x,y]];\mathsf{inr}(w).\,E^*[?(w)])\\
&\to\ \mathsf{case}\,u^*\,\mathsf{of}\,(\mathsf{inl}(z).\,E^*[t^*[\mathsf{fst}(z),\mathsf{snd}(z)/x,y]];\mathsf{inr}(w).\,?(w))\\
&=\ \mathsf{case}\,u^*\,\mathsf{of}\,(\mathsf{inl}(z).\,E^*[t^*][\mathsf{fst}(z),\mathsf{snd}(z)/x,y];\mathsf{inr}(w).\,?(w))\\
&=\ \mathsf{case}\,u^*\,\mathsf{of}\,(\mathsf{inl}(z).\,(E[t])^*[\mathsf{fst}(z),\mathsf{snd}(z)/x,y];\mathsf{inr}(w).\,?(w))\\
&=\ (\mathsf{let}\ !x\otimes y\ \mathsf{be}\ u\ \mathsf{in}\ E[t])^*\ .
\end{aligned}$$

The subcase in which $E[-]$ is $\mathsf{let}\ !x_0\otimes y_0\ \mathsf{be}\ [-]\ \mathsf{in}\ t_0$ is calculated separately by:

$$\begin{aligned}
&(\mathsf{let}\ !x_0\otimes y_0\ \mathsf{be}\ (\mathsf{let}\ !x\otimes y\ \mathsf{be}\ u\ \mathsf{in}\ t)\ \mathsf{in}\ t_0)^*\\
&\quad=\ \mathsf{case}\,(\mathsf{case}\,u^*\,\mathsf{of}\,(\mathsf{inl}(z).\,t^*[\mathsf{fst}(z),\mathsf{snd}(z)/x,y];\mathsf{inr}(w).\,?(w)))\,\mathsf{of}\\
&\qquad(\,\mathsf{inl}(z_0).\,t_0^*[\mathsf{fst}(z_0),\mathsf{snd}(z_0)/x_0,y_0];\ \mathsf{inr}(w_0).\,?(w_0)\,)\\
&\quad\to\ \mathsf{case}\,u^*\,\mathsf{of}\\
&\qquad(\,\mathsf{inl}(z).\,\mathsf{case}\,(t^*[\mathsf{fst}(z),\mathsf{snd}(z)/x,y])\,\mathsf{of}\,(\mathsf{inl}(z_0).\,t_0^*[\mathsf{fst}(z_0),\mathsf{snd}(z_0)/x_0,y_0];\mathsf{inr}(w_0).\,?(w_0));\\
&\qquad\ \mathsf{inr}(w).\,\mathsf{case}\,(?(w))\,\mathsf{of}\,(\mathsf{inl}(z_0).\,t_0^*[\mathsf{fst}(z_0),\mathsf{snd}(z_0)/x_0,y_0];\mathsf{inr}(w_0).\,?(w_0))\,)\\
&\quad\to\ \mathsf{case}\,u^*\,\mathsf{of}\\
&\qquad(\,\mathsf{inl}(z).\,\mathsf{case}\,(t^*[\mathsf{fst}(z),\mathsf{snd}(z)/x,y])\,\mathsf{of}\,(\mathsf{inl}(z_0).\,t_0^*[\mathsf{fst}(z_0),\mathsf{snd}(z_0)/x_0,y_0];\mathsf{inr}(w_0).\,?(w_0));\\
&\qquad\ \mathsf{inr}(w).\,?(w)\,)\\
&\quad=\ \mathsf{case}\,u^*\,\mathsf{of}\\
&\qquad(\,\mathsf{inl}(z).\,(\mathsf{case}\,t^*\,\mathsf{of}\,(\mathsf{inl}(z_0).\,t_0^*[\mathsf{fst}(z_0),\mathsf{snd}(z_0)/x_0,y_0];\mathsf{inr}(w_0).\,?(w_0)))[\mathsf{fst}(z),\mathsf{snd}(z)/x,y];\\
&\qquad\ \mathsf{inr}(w).\,?(w)\,)\\
&\quad=\ (\mathsf{let}\ !x\otimes y\ \mathsf{be}\ u\ \mathsf{in}\ \mathsf{let}\ !x_0\otimes y_0\ \mathsf{be}\ t\ \mathsf{in}\ t_0)^*\ .
\end{aligned}$$

□

Proposition B.5 (Strong normalization for EEC+)**.** *If* $\Gamma \mid \Delta \vdash_\Sigma t\colon \mathsf{A}$ *in* EEC+ *then* t *is strongly normalizing under* $\to$*.*

Proof. Suppose there is an infinite reduction sequence from t. By Lemmas B.3 and B.4 respectively, $\Gamma^*, \Delta^* \vdash_{\Sigma^*} t^*\colon \mathsf{A}^*$ in $\lambda+$ and t^* has an infinite reduction sequence. This contradicts strong normalization for $\lambda+$. □

We make two remarks about alternative proof techniques. Instead of proving strong normalization for EEC+ via a reduction to $\lambda+$, it is not difficult to give a direct proof for EEC+. We have carried out such a proof using the method of [23]. Also, for the application in Section B.2 below, weak normalization suffices, which can be proved by the usual simple inductive argument [33]. Nevertheless, it seems worthwhile to establish the stronger result, since true, and the approach followed in this section, of reducing strong normalization to a known result, has the benefit of conciseness.

B.2 Proof of syntactic conservativity

We now prove Theorem 4.3. As in the statement of that theorem, we let (X, Y) be one of the following pairs of calculi, (EC, EC+), (EC, EEC), (EC, EEC+), (EC+, EEC+), (EEC, EEC+). Also, we assume the signature Σ contains only types from the calculus X.

The strategy for the establishing the syntactic conservativity of Y over X is to show that any normal form term in system Y whose type is in X is also a system X term. Since every system Y term is normalizable and equal to its normal form, the theorem follows.

To do this, it is useful to characterise the normal forms, relative to the rewrite relation $\to$, using a grammar.

$$v ::= a \mid \star \mid \langle v, v\rangle \mid \lambda x\colon \mathsf{A}.\, v \mid\, !v \mid \mathsf{let}\, !x\, \mathsf{be}\, a\, \mathsf{in}\, v \mid ?(a) \mid \mathsf{inl}(v) \mid \mathsf{inr}(v) \mid \mathsf{case}\, a\, \mathsf{of}\, (\mathsf{inl}(x).\, v; \mathsf{inr}(y).\, v) \mid$$
$$\lambda^{\circ} x\colon \underline{\mathsf{A}}.\, v \mid\, !v \otimes v \mid \mathsf{let}\, !x \otimes y\, \mathsf{be}\, a\, \mathsf{in}\, v \mid \underline{?}(a) \mid \underline{\mathsf{inl}}(v) \mid \underline{\mathsf{inr}}(v) \mid \underline{\mathsf{case}}\, a\, \mathsf{of}\, (\underline{\mathsf{inl}}(x).\, v; \underline{\mathsf{inr}}(y).\, v)$$

$$a ::= x \mid \mathsf{fst}(a) \mid \mathsf{snd}(a) \mid a\,(v) \mid a[v] \mid f(v, \ldots, v) \mid g(v, \ldots, v \mid v)$$

Lemma B.6. *If* $\Gamma \mid \Delta \vdash_{\Sigma} t\colon \mathsf{A}$ *in* EEC+ *and* t *is in normal form then* t *satisfies the grammar for terms* v*.*

The converse holds too, but we will not prove this since we have no need for the result.

Proof. We prove that for any term t, if t is a normal form, then t is a v-term. The proof is a straightforward induction over the structure of t, and we give just one illustrative case.

If t is a normal form of the form $\mathsf{case}\, s\, \mathsf{of}\, (\mathsf{inl}(x).\, u_1; \mathsf{inr}(y).\, u_2)$ then s, u_1, u_2 must be normal forms and so, by induction hypothesis, must be v-terms. The v-term s has type $\mathsf{A} + \mathsf{B}$, and so, for type reasons, must be one of $\mathsf{inl}(v)$, $\mathsf{inr}(v)$, $?(a)$, $\mathsf{case}\, a\, \mathsf{of}\, (\mathsf{inl}(x).\, v; \mathsf{inr}(y).\, v)$, or a. (Note that s cannot be a term of the form $\mathsf{let}\, !x\, \mathsf{be}\, a\, \mathsf{in}\, v$, $\mathsf{let}\, !x \otimes y\, \mathsf{be}\, a\, \mathsf{in}\, v$, $\underline{\mathsf{case}}\, a\, \mathsf{of}\, (\underline{\mathsf{inl}}(x).\, v; \underline{\mathsf{inr}}(y).\, v)$ or $\underline{?}(a)$ because their types are required to be computation types.) However, if s were one of $\mathsf{inl}(v)$, $\mathsf{inr}(v)$, $?(a)$, $\mathsf{case}\, a\, \mathsf{of}\, (\mathsf{inl}(x).\, v; \mathsf{inr}(y).\, v)$ then t would be a redex, contradicting its normality. Thus s must be of the form a. Whence t is indeed a v-term.

The arguments for the other cases are similar. □

Lemma B.7. *Let* (X, Y) *be as specified above, and let* $\Gamma \mid \Delta$ *be a context whose types are all in system* X*. Then*

1. *If* $\Gamma \mid \Delta \vdash_{\Sigma} a\colon \mathsf{B}$ *in system* Y *then* B *is a system* X *type and* $\Gamma \mid \Delta \vdash_{\Sigma} a\colon \mathsf{B}$ *in system* X*.*
2. *If* $\Gamma \mid \Delta \vdash_{\Sigma} v\colon \mathsf{B}$ *in system* Y *and* B *is a system* X *type then* $\Gamma \mid \Delta \vdash_{\Sigma} v\colon \mathsf{B}$ *in system* X*.*

(Here a and v range over terms specified by the grammar above.)

Proof. We assume, without loss of generality, that system Y is EEC+. The two statements of the lemma are proved by simultaneous induction over the structure of a and v. For statement 1, we consider two cases.

Suppose that $\Gamma \mid \Delta \vdash_{\Sigma} a[v]\colon \underline{\mathsf{B}}$ in EEC+. Then $\Gamma \mid - \vdash a\colon \underline{\mathsf{A}} \multimap \underline{\mathsf{B}}\Sigma$ and $\Gamma \mid \Delta \vdash v\colon \underline{\mathsf{A}}\Sigma$ in EEC+, for some computation type $\underline{\mathsf{A}}$. By induction hypothesis 1, we have that $\underline{\mathsf{A}} \multimap \underline{\mathsf{B}}$ is a system X type (hence X is EEC), and $\Gamma \mid - \vdash a\colon \underline{\mathsf{A}} \multimap \underline{\mathsf{B}}\Sigma$ in X. Hence $\underline{\mathsf{A}}$ is a system X type, and, by induction hypothesis 2, also $\Gamma \mid \Delta \vdash_{\Sigma} v\colon \underline{\mathsf{A}}$ in X. Thus indeed $\Gamma \mid \Delta \vdash_{\Sigma} a[v]\colon \underline{\mathsf{B}}$ in X.

Suppose that $\Gamma \mid \Delta \vdash_{\Sigma} f(v_1, \ldots, v_k \mid v_{k+1})\colon \underline{\mathsf{C}}$ in EEC+. Since we have assumed that the signature Σ contains only types from X, we have $g\colon (\mathsf{A}_1, \ldots, \mathsf{A}_k \mid \underline{\mathsf{B}}) \to \underline{\mathsf{C}} \in \Sigma$ where each of $\mathsf{A}_1, \ldots, \mathsf{A}_k$, $\underline{\mathsf{B}}$, $\underline{\mathsf{C}}$ are system X types. Since $\Gamma \mid - \vdash v_1\colon \mathsf{A}_1\Sigma$, …, $\Gamma \mid - \vdash v_k\colon \mathsf{A}_k\Sigma$ and $\Gamma \mid \Delta \vdash_{\Sigma} v_{k+1}\colon \underline{\mathsf{B}}$ in EEC+, by induction hypothesis 2, the same typing judgments are derivable in system X. Thus indeed $\Gamma \mid \Delta \vdash_{\Sigma} f(v_1, \ldots, v_k \mid v_{k+1})\colon \underline{\mathsf{C}}$ in X.

For statement 2, the case that $v = a$ is already covered by statement 1. We consider just one of the sixteen remaining cases.

Suppose that $\Gamma \mid \Delta \vdash_\Sigma \mathsf{let}\, !x \otimes y\, \mathsf{be}\, a\, \mathsf{in}\, v \colon \underline{\mathsf{C}}$ in EEC+, where $\underline{\mathsf{C}}$ is a system X type. Then $\Gamma \mid \Delta \vdash_\Sigma a \colon !\mathsf{A} \otimes \underline{\mathsf{B}}$ and $\Gamma, x \colon \mathsf{A} \mid y \colon \underline{\mathsf{B}} \vdash_\Sigma v \colon \underline{\mathsf{C}}$ in EEC+. By induction hypothesis 1, we have that $!\mathsf{A} \otimes \underline{\mathsf{B}}$ is a system X type (hence X is EEC) and $\Gamma \mid \Delta \vdash_\Sigma a \colon !\mathsf{A} \otimes \underline{\mathsf{B}}$ in system X. By induction hypothesis 2, we have $\Gamma, x \colon \mathsf{A} \mid y \colon \underline{\mathsf{B}} \vdash_\Sigma v \colon \underline{\mathsf{C}}$ in X. Thus $\Gamma \mid \Delta \vdash_\Sigma \mathsf{let}\, !x \otimes y\, \mathsf{be}\, a\, \mathsf{in}\, v \colon \underline{\mathsf{C}}$ in X as required. □

Finally we complete the proof of Theorem 4.3.

Proof of Theorem 4.3. Suppose $\Gamma \mid \Delta \vdash_\Sigma u \colon \mathsf{A}$ in system Y and suppose all the types occurring in Γ, Δ and A are in system X. By Proposition B.5, u reduces to some normal form t in finitely many steps. By Lemma B.1, $\Gamma \mid \Delta \vdash_\Sigma u = t \colon \mathsf{A}$ in system Y. Moreover, by Lemma B.6.2, t is a v-term, and so by Lemma B.7, $\Gamma \mid \Delta \vdash_\Sigma t \colon \mathsf{A}$ in system X. □

References

[1] A. Barber. *Linear Type Theories, Semantics and Action Calculi*. PhD thesis, Department of Computer Science, 1997.

[2] P. N. Benton. A mixed linear and non-linear logic: Proofs, terms and models. In *Proc. Computer Science Logic (CSL) 1994*, volume 933 of *LNCS*. Springer, 1995.

[3] P. N. Benton and P. Wadler. Linear logic, monads, and the lambda calculus. In *Proc. 11th Annual Symposium on Logic in Computer Science (LICS)*, 1996.

[4] J. Berdine, P. W. O'Hearn, U. Reddy, and H. Thielecke. Linear continuation-passing. *Higher Order and Symbolic Computation*, 15:181–208, 2002.

[5] M. Churchill and J. Laird. A logic of sequentiality. In *Proc. Computer Science Logic*, volume 6247 of *Lecture Notes in Computer Science*, pages 215–229. Springer Verlag, 2010.

[6] J. Egger, R. E. Møgelberg, and A. Simpson. Enriching an effect calculus with linear types. In *Proc. Computer Science Logic (CSL)*, volume 5771 of *LNCS*. Springer, 2009.

[7] J. Egger, R. E. Møgelberg, and A. Simpson. Linearly-used continuations in the enriched effect calculus. In *Proc. Foundations of Software Science and Computation Structures (FoSSaCS)*, volume 6014 of *LNCS*. Springer, 2010.

[8] J. Egger, R. E. Møgelberg, and A. Simpson. Categorical models for the enriched effect calculus, 2012. In preparation.

[9] J. Egger, R. E. Møgelberg, and A. Simpson. Linear-use CPS translations in the enriched effect calculus. *Logical Methods in Computer Science*, to appear, 2012.

[10] A. Filinski. *Controlling Effects*. PhD thesis, Carnegie Mellon University, 1996.

[11] J.-Y. Girard. Linear logic. *Theoretical Computer Science*, 50:1–102, 1987.

[12] J.-Y. Girard. *Proofs and Types*, volume 7 of *Cambridge Tracts in Theoretical Computer Science*. Cambridge University Press, 1989.

[13] J.-Y. Girard. A new constructive logic: classical logic. *Mathematical Structures in Computer Science*, 1:255–296, 1991.

[14] 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.

[15] F. Joachimski and R. Matthes. Short proofs of normalization for the simply-typed lambda-calculus, permutative conversions and Gödel's T. *Archive for Mathematical Logic*, 42:59–87, 2003.

[16] A. Joyal. Free bicomplete categories. *C. R. Math. Rep. Acad. Sci. Canada*, 17:219–224, 1995.

[17] G. M. Kelly. *Basic Concepts of Enriched Category Theory*, volume 64 of *LMS Lecture Notes*. Cambridge University Press, 1982.

[18] A. Kock. Strong functors and monoidal monads. *Archiv der Mathematik*, 21:1–10, 1970.

[19] J. Laird. A categorical semantics of higher-order store. In *Proceedings of CTCS '02*, number 69 in ENTCS. Elsevier, 2002.

[20] S. Mac Lane. *Categories for the Working Mathematician*. Springer, 1998. Second edition.

[21] P. B. Levy. *Call-by-push-value. A functional/imperative synthesis*. Semantic Structures in Computation. Springer, 2004.

[22] P. B. Levy. Adjunction models for call-by-push-value with stacks. *Theory and Applications of Categories*, 14:75–110, 2005.

[23] S. Lindley and I. Stark. Reducibility and TT-lifting for computation types. In *Proceedings of TLCA 2005*, number 3461 in LNCS, 2005.

[24] F. E. J. Linton. Coequalizers in categories of algebras. In *Seminar on Triples and Categorical Homology Theory*, volume 80 of *Lecture Notes in. Math*, pages 75–90. Springer-Verlag, 1969.

[25] R. E. Møgelberg and A. Simpson. Relational parametricity for control considered as a computational effect. In *Proc. Twenty-third Annual Conference on Mathematical Foundations of Programming Semantics (MFPS XXIII)*, volume 173, pages 295–312, 2007.

[26] R. E. Møgelberg and A. Simpson. A logic for parametric polymorphism with effects. In *Post-conference proceedings for selected papers from TYPES 2007*, volume 4941 of *LNCS*, pages 142–156. Springer, 2008.

[27] R. E. Møgelberg and A. Simpson. Relational parametricity for computational effects. *Logical Methods in Computer Science*, 5(3:7), 2009.

[28] R. E. Møgelberg and S. Staton. Linearly-used state in models of call-by-value. In *Proc. 4th Conference on Algebra and Coalgebra in Computer Science (CALCO)*, volume 6859 of *LNCS*, pages 293–313. Springer, 2011.

[29] E. Moggi. Computational lambda-calculus and monads. In *Proc. 4th Annual Symposium on Logic in Computer Science (LICS)*, pages 14–23, 1989.

[30] E. Moggi. Notions of computation and monads. *Information and Computation*, 93:55–92, 1991.

[31] P. W. O'Hearn and J. C. Reynolds. From Algol to polymorphic linear lambda-calculus. *Journal of the ACM*, 47:167–223, 2000.

[32] G. Plotkin and A. J. Power. Computational effects and operations: An overview. *Electr. Notes Theor. Comput. Sci.*, 73:149–163, 2004.

[33] D. Prawitz. *Natural Deduction — A proof theoretical study.* Almquist and Wiksell, Stockholm, 1965.

[34] D. Prawitz. Ideas and results in proof theory. In *Proceedings of the second Scandinavian logic symposium*, pages 237–309. North-Holland, 1971.

[35] A. Simpson. *The Proof Theory and Semantics of Intuitionistic Modal Logic.* PhD thesis, University of Edinburgh, 1994.