Context dependence and coeffect-delimited comonads

Dominic Orchard, Tomas Petricek

University of Cambridge, UK

Abstract. Recently, the notion of effect-delimited monads has been proposed, unifying monadic denotational semantics of effects with traditional effect systems. Independently, the dual notion of coeffect-delimited comonads (termed *indexed comonads* in the literature) have been used to give a semantics to contextual computations (such as implicit parameters and dataflow). This dual situation however requires extra structure; the dualisation is asymmetric since contexts in the lambda-calculus are structured. In this paper, we derive coeffect-delimited comonads and generalise the derivation to additional monoidal structures. These are used 1) to give the semantics of a contextual lambda calculus and motivate the axioms of coeffect systems 2) to extend effects with delimited monoidal operations which model non-deterministic choice and parallelism.

1 Introduction

Monads are widely used in computer science to abstract the composition of effectful, or impure, computations [6]. In this scheme, denotations (or functions) are of the form $A \to \mathsf{T}B$, where an object mapping (or type constructor) T encodes the effect *produced* by the computation. Comonads, the dual of monads, are used to abstract composition of context-dependent computations [14,9], with denotations/functions of the form $\mathsf{D}A \to B$, where D encodes the context *consumed* by the computation. In this way, monads and comonads can be seen as abstractions over *impurity*, in the output and input respectively.

Recently, the notion of an *effect-delimited monad* has been proposed which unifies monadic denotational semantics with *effect systems* [4]. Delimited monads abstract composition for morphisms of the form $A \to \mathsf{T}_F B$, where F ranges over effect annotations with a monoid structure describing sequential composition in an effect system. The abstract structure of effect-delimited monads arises neatly as a homomorphism between a monoid of effect annotations and the monoid of endofunctor composition, *i.e.*, they are lax monoidal functors, $\mathsf{T} : F \to [\mathcal{C}, \mathcal{C}]$.

For the dual situation of context dependence, the general notion of a *coeffect system* has been recently proposed, dualising effect systems [12]. *Indexed comonads* where proposed to unify a categorical semantics of a context-dependent (contextual) λ-calculus (called the *coeffect calculus*) with an abstract coeffect system. Indexed comonads happen to be the dual of effect-delimited monads.

This paper develops semantic foundations of context dependence using delimited (indexed) categorical structures. We make the following contributions:

– Coeffect-delimited comonads are derived (with examples) (§2), arriving at the previously proposed notion of indexed comonads. This derivation dualises that of Katsumata's effect-delimited monads.
– The derivation of delimited (co)monads is generalised to delimited monoidal functors (§3) required for a delimited comonadic semantics of the λ-calculus.
– The delimited structures of the previous two sections are used to give a delimited semantics of a contextual λ-calculus with a general coeffect system (§4), elucidating the design and axioms of the coeffect calculus [12].
– Delimited monoidal functors are applied to effectful computations to model non-deterministic choice, failure, and parallelism (§5), with corresponding effect annotations.

We begin with an introduction to coeffect systems and comonads.

Notation We write categories in calligraphy (*e.g.*, $\mathcal{C}, \mathcal{I}$), functors in sans (*e.g.*, F, T), objects in uppercase (*e.g.*, A, B, X, Y), and σ, τ range over types. Monadic and comonadic functors are written T and D by convention. For a category $\mathcal{C}$, its class of objects is written $|\mathcal{C}|$ and its hom-set of morphisms $X \to Y$ is written $\mathcal{C}(X, Y)$. Natural transformations are written $\eta : \mathsf{F} \dot{\to} \mathsf{G}$ or $\eta_A : \mathsf{F}A \to \mathsf{G}A$.

An example coeffect system Consider the simply-typed λ-calculus extended with a notion of *implicit parameters* (similar to the implicit parameter feature of Haskell [5]). Implicit parameters are dynamically-scoped variables, rather than lexically-scoped like usual variables in the λ-calculus. The syntax and types are:

$$e ::= v \mid \lambda v.e \mid e\, e \mid ?v \mid e \textbf{ with } ?v = e \qquad \tau ::= T \mid \tau \xrightarrow{P} \tau$$

where T ranges over type constants, $?v$ are implicit parameters, the **with** construct dynamically binds implicit parameters, and P ranges over sets of implicit-parameter/type pairs, written $\{?v_1 : \tau_1, \ldots, ?v_n : \tau_n\}$. Thus, function types are annotated with the implicit parameter requirements of the function body.

This implicit parameter calculus has a *type-and-coeffect system* with judgments $\Gamma \,?\, P \vdash e : \tau$ meaning e is an expression, of type τ with free-variable typing assumptions Γ, that may use implicit parameters in the set P (the coeffect).

Type-and-coeffect rules are shown in Figure 1. The (?var) and (with) rule introduce and eliminate implicit parameter requirements respectively. The (with) rule erases coeffect information from types (via "\") which explains the semantics of **with**-binding as the dynamic binding of all occurrences of an implicit parameter in the dynamic scope (including occurrences inside λ-abstractions).

Note, we avoid α-renaming for implicit paramters and capture-avoiding substitution by requiring freshness for all implicit parameter names (where the side-condition of (with) means $?v$ is fresh for the bound term e' and its type σ).

Example 1. The following term **with**-binds $?x$, satisfying the *latent* implicit parameter requirement for $?x$ in f, thus erasing the coeffect $\{?x : \mathbb{N}\}$ in the type:

$$\textbf{(with)}\ \frac{\textbf{(var)}\ \dfrac{}{f : \mathbb{N} \xrightarrow{?x:\mathbb{N}} \mathbb{N} \,?\, \emptyset \vdash f : \mathbb{N} \xrightarrow{?x:\mathbb{N}} \mathbb{N}} \qquad \textbf{(const)}\ \dfrac{}{f : \mathbb{N} \xrightarrow{?x:\mathbb{N}} \mathbb{N} \,?\, \emptyset \vdash 42 : \mathbb{N}}}{f : \mathbb{N} \xrightarrow{?x:\mathbb{N}} \mathbb{N} \,?\, \emptyset \vdash f \textbf{ with } ?x = 42 : \mathbb{N} \xrightarrow{\emptyset} \mathbb{N}}$$

$$\text{(var)}\frac{v : \tau \in \Gamma}{\Gamma \,?\, \emptyset \vdash v : \tau} \qquad \text{(?var)}\frac{}{\Gamma \,?\, \{?v : \tau\} \vdash ?v : \tau} \qquad \text{(const)}\frac{}{\Gamma \,?\, \emptyset \vdash c : T}$$

$$\text{(abs)}\frac{\Gamma, v : \sigma \,?\, P_1 \uplus P_2 \vdash e : \tau}{\Gamma \,?\, P_1 \vdash \lambda v.e : \sigma \xrightarrow{P_2} \tau} \qquad \text{(app)}\frac{\Gamma \,?\, P_1 \vdash e : \sigma \xrightarrow{P_2} \tau \quad \Gamma \,?\, P_3 \vdash e' : \sigma}{\Gamma \,?\, P_1 \cup P_2 \cup P_3 \vdash e\, e' : \tau}$$

$$\text{(with)}\frac{\Gamma \,?\, P \vdash e : \tau \quad \Gamma \,?\, P' \vdash e' : \sigma}{\Gamma \,?\, (P - \{?v : \sigma\}) \cup P' \vdash e \textbf{ with } ?v = e' : (\tau \backslash \{?v : \sigma\})} \; ?v\#(e', \sigma)$$

Fig. 1. Type-and-coeffect system for implicit parameters

The (abs) rule is non-inductive in its coeffects. Reading the rule top down, requirements of the function body are split between the declaration site (immediate coeffects) and call site (latent coeffects). Reading bottom up, the implicit paramters supplied at the declaration and call site are combined using disjoint union for the function body (rather than union as in the other rules). Since disjoint union is non-injective, (abs) introduces a source of non-determinism. However, **with**-binding restores determinism since partially-applied disjoint union *is* injective. A deterministic implementation follows by restricting the top-level expression to the $\emptyset$ coeffect, *i.e.*, all implicit parameters are **with**-bound.

Comonads Comonads structure the semantics of various context-dependent (contextual) notions of computation [14,9] modelled as morphisms of the form $\mathsf{D}A \to B$ where D encodes additional context over A values.

Definition 1. *A* comonad *comprises an endofunctor $D : \mathcal{C} \to \mathcal{C}$ and two natural transformations $\varepsilon_A : \mathsf{D}A \to A$ (counit) and $\delta_A : \mathsf{D}A \to \mathsf{DD}A$ (comultiplication), satisfying conditions* [C1] $\varepsilon\mathsf{D} \circ \delta = id$, [C2] $\mathsf{D}\varepsilon \circ \delta = id$, *and* [C3] $\delta\mathsf{D} \circ \delta = \mathsf{D}\delta \circ \delta$.

An alternate definition replaces δ by *extension*, which maps morphisms $k : \mathsf{D}A \to B$ to $k^\dagger : \mathsf{D}A \to \mathsf{D}B$ such that $k^\dagger = \mathsf{D}k \circ \delta$. Analogous laws hold following this equivalence between comultiplication and extension. Extension also subsumes functoriality of D where $\mathsf{D}f = (f \circ \varepsilon)^\dagger$ thus D need only be an object mapping.

Composition for some $f : \mathsf{D}A \to B, g : \mathsf{D}B \to C$ is then $g \mathbin{\hat{\circ}} f = g \circ f^\dagger$. By the conditions [C1-3], this composition is associative and has ε as the identity. This forms the *coKleisli category* of a comonad, where $CoK(\mathcal{C})(A, B) = \mathcal{C}(\mathsf{D}A, B)$.

Example 2 (Product comonad). For some fixed object X, the object mapping $\mathsf{D}A = A \times X$ is a comonad with operations:

$$\varepsilon(a, x) \mapsto a \qquad f^\dagger(a, x) \mapsto (f(a, x), x)$$

This comonad models the "plumbing" of additional values throughout a computation, *e.g.*, given $f : A \times X \to B$ and $g : B \times X \to C$ their composition computes $(g \mathbin{\hat{\circ}} f)(a, x) \mapsto g(f(a, x), x)$, thus x is passed through to each subcomputation.

Although the product comonad is rather prosaic, it is a useful example as we generalise to delimited comonads (§2). There are many other interesting comonads including streams, functions, arrays, and trees (see, *e.g.*, [14,10,9]). We focus instead on examples of delimited comonads in this paper.

Semantics A categorical, comonadic semantics for a (typed) contextual calculus has denotations $[\![\Gamma \vdash e : \tau]\!] : \mathsf{D}[\![\Gamma]\!] \to [\![\tau]\!]$ in $\mathcal{C}$, with a suitable interpretation for Γ (usually products). In this semantics, the comonad provides sequential composition of contextual computations. For a contextual λ-calculus, the additional structure of a *monoidal comonad* is needed for the semantics of abstraction [14] (discussed further in §3 and §4). Throughout we omit $[\![-]\!]$ on objects for brevity.

Uniform context propagation The product comonad provides a semantics for implicit parameters where the set of implicit parameters is *uniform* throughout a computation, *i.e.*, where subcomputations cannot have different requirements. This relates to a general property of comonads, called *shape preservation* [10]: for any morphism $f : \mathsf{D}A \to B$ then $\mathsf{D}!_B \circ f^\dagger = \mathsf{D}!_A$ where $\mathsf{D}!_A$ computes the shape of the context by erasing its elements. Thus, for any comonad, sequential composition propagates a context of uniform shape, fixed by the incoming $\mathsf{D}A$ value. However, many notions of contextual computation have subcomputations with differing contextual requirements, as in the implicit parameter example.

Consider denotations $f : A \times X \to B$ and $g : B \times Y \to C$ which require an X and Y parameter respectively. These can compose by merging requirements such that $g \mathbin{\hat{\circ}} f : A \times (X \times Y) \to C$. Such a composition does not arise from a comonad, where context is uniform throughout, but is instead provided by the notion of a *coeffect-delimited comonad*.

2 Coeffect-delimited comonads

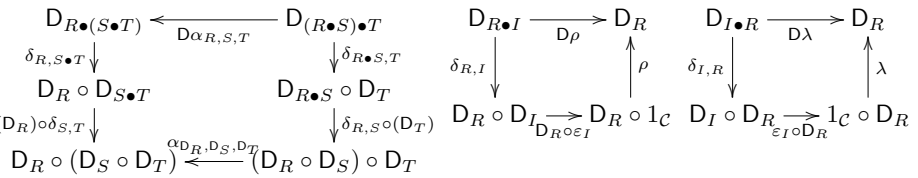
Katsumata recently introduced effect-delimited monads [4]. We present the dual derivation for coeffect-delimited comonads, which was not given previously for indexed comonads; they were more ad hoc [12]. The phrase *delimited* refers to defining the range/extent of the captured (co)effects (the upper bound).

Definition 2. *A* coeffect-delimited *comonad, for a category $\mathcal{C}$, comprises a strict monoidal category $(\mathcal{I}, \bullet, I)$ and a functor $\mathsf{D} : \mathcal{I} \to [\mathcal{C}, \mathcal{C}]$ which is a* colax monoidal *functor between the strict monoid on $\mathcal{I}$ and the strict monoid of endofunctor composition in $[\mathcal{C}, \mathcal{C}]$, and therefore has natural transformations:*

$$\delta_{X,Y} : \mathsf{D}(X \bullet Y) \dot{\to} \mathsf{D}X \circ \mathsf{D}Y \qquad \varepsilon_I : \mathsf{D}I \dot{\to} 1_{\mathcal{C}}$$

where $1_{\mathcal{C}}$ is the identity endofunctor on $\mathcal{C}$ and $\circ$ is endofunctor composition.

Expanding this definition, the colax monoidal part of D provides a homomorphism from $(\mathcal{I}, \bullet, I)$ to $([\mathcal{C}, \mathcal{C}], \circ, 1_{\mathcal{C}})$. The standard coherence conditions of colax monoidal functors for D are then (where from now on $\mathsf{D}A$ is written D_A):



where α, λ, and ρ are natural isomorphisms for associativity and unitality in each monoidal category. Since both monoids are strict, α, λ, ρ are identities, *e.g.*, $\mathsf{D}_{I \bullet R} = \mathsf{D}_R = \mathsf{D}_{R \bullet I}$, $\mathsf{F} \circ I_{\mathcal{C}} = \mathsf{F} = I_{\mathcal{C}} \circ \mathsf{F}$. The colax monoidal conditions therefore collapse to:

$$\begin{array}{ccc} \mathsf{D}_R & \xrightarrow{\delta_{R,I}} & \mathsf{D}_R\mathsf{D}_I \\ {\scriptstyle \delta_{I,R}}\downarrow & \overset{\text{[C2]}}{\searrow\!\searrow} & \downarrow{\scriptstyle \mathsf{D}_R\varepsilon_I} \\ \mathsf{D}_I\mathsf{D}_R & \xrightarrow[\varepsilon_I \mathsf{D}_R]{\text{[C1]}} & \mathsf{D}_R \end{array} \qquad \begin{array}{ccc} \mathsf{D}_{R \bullet S \bullet T} & \xrightarrow{\delta_{R \bullet S, T}} & \mathsf{D}_{R \bullet S}\mathsf{D}_T \\ {\scriptstyle \delta_{R, S \bullet T}}\downarrow & \text{[C3]} & \downarrow{\scriptstyle \delta_{R,S}\mathsf{D}_T} \\ \mathsf{D}_R\mathsf{D}_{S \bullet T} & \xrightarrow[\mathsf{D}_R \delta_{S,T}]{} & \mathsf{D}_R\mathsf{D}_S\mathsf{D}_T \end{array} \tag{1}$$

These correspond to delimited analogues of the usual comonad laws [C1-3].

Example 3 (Delimited product comonad). Given a set X, with the monoidal category of X-subsets $(\mathcal{P}(X), \cup, \emptyset)$, then $\mathsf{D} : \mathcal{P}(X) \to [\mathcal{C}, \mathcal{C}]$ where $\mathsf{D}_X A = A \times X$ and $\mathsf{D}_X f = f \times id_X$ is a delimited comonad with:

$$\varepsilon_\emptyset\,(a, \emptyset) \mapsto a \qquad \delta_{F,G}(a, x) \mapsto ((a, x - (G - F)), x - (F - G))$$

where $x : F \cup G$ and thus the F-only subset of x is $x - (G - F)$ and the G-only subset of x is $x - (F - G)$ (where $(-)$ is set difference).

The delimited product comonad gives a semantics for composing computations with additional information/resources provided by the (local or global) execution context. This includes the implicit parameters example, where X is a (nominal) set of possible variable-type pairs. The parameter of D then corresponds to coeffect annotations, where composition (expressed using **let**-binding) has semantics:

$$\frac{[\![\Gamma \,?\, S \vdash e : \sigma]\!] = k : \mathsf{D}_S[\![\Gamma]\!] \to [\![\sigma]\!] \qquad [\![x : \sigma \,?\, R \vdash e' : \tau]\!] = k' : \mathsf{D}_R[\![\sigma]\!] \to [\![\tau]\!]}{[\![\Gamma \,?\, R \cup S \vdash \mathbf{let}\, x = e\, \mathbf{in}\, e' : \tau]\!] = k' \circ \mathsf{D}_R k \circ \delta_{R,S} : \mathsf{D}_{(R \cup S)}[\![\Gamma]\!] \to [\![\tau]\!]}$$

Note that **let** here is specialised where the inner term e' has a single free-variable, thus only composition is needed without any "plumbing" of contexts. Additional structure is needed for a semantics of abstraction and application (§3).

As with comonads, delimited comonads have an *extension* form where for all $f : \mathsf{D}_Y A \to B$ then $f^\dagger_{X,Y} : \mathsf{D}_{X \bullet Y} A \to \mathsf{D}_X B$ is the delimited extension such that $f^\dagger_{X,Y} = \mathsf{D}_X f \circ \delta_{X,Y}$. Extension is used for brevity in later sections.

Example 4 (Delimited partiality comonad). The boolean conjunction monoid $(\mathbb{B}, \wedge, \top)$ describes partiality, where $\bot$ and $\top$ indicate undefined and defined computations respectively. A delimited comonad $\mathsf{D} : \mathbb{B} \to [\mathcal{C}, \mathcal{C}]$ is defined:

$$\begin{array}{ll} \mathsf{D}\,\bot A = 1 & k^\dagger_{\top,\top} a \mapsto k\,a \quad k^\dagger_{\top,\bot} 1 \mapsto k\,1 \\ \mathsf{D}\,\top A = A & k^\dagger_{\bot,\bot} 1 \mapsto 1 \quad k^\dagger_{\bot,\top} 1 \mapsto 1 \end{array} \qquad \varepsilon_\top\, a \mapsto a$$

This delimited comonad encodes the notion of a partial context/input to a computation, and has been previously shown to give a simple liveness analysis coupled with a semantics where composition using the above extension encodes dead-code elimination [12]. Here D is essentially a dependently-typed partiality functor $\mathsf{D}A = 1 + A$ where indices tag the coproduct. Note however, that $\mathsf{D}A = 1 + A$ is *not* a comonad since $\varepsilon : \mathsf{D}A \to A$ is not defined for all inputs.

Example 5 (Bounded reuse). In Girard *et al.*'s bounded linear logic (BLL) the *bounded reuse* of a proposition A is written $!_X A$, meaning it may be reused at most X times [3]. A model for $!_X A$ is suggested as the X-times tensor of A, *e.g.*, $!_X A = A \times \ldots \times A$ (or, $!_X A = A^X$). The *storage axiom* of BLL corresponds roughly to a delimited extension operation of a delimited comonad for !:

$$\text{Storage}\frac{!_{\bar{y}}\Gamma \vdash A}{!_{x\bar{y}}\Gamma \vdash !_x A}$$

Note that $\bar{y}$ is a vector with $\bar{y} = y_1, \ldots, y_n$ for $|\Gamma| = n$. We return to this later (§6). Bounded reuse is intuitively modelled by a delimited comonad $! : \mathbb{N} \to [\mathcal{C}, \mathcal{C}]$, with $!_X A = A^X$, the natural-number multiplication monoid $(\mathbb{N}, \times, 1)$, and:

$$f^{\dagger}_{X,Y}\langle a_1, .., a_{XY}\rangle \mapsto \langle f\langle a_1, .., a_Y\rangle, f\langle a_{Y+1}, .., a_{Y+Y}\rangle, .., f\langle a_{(X-1)Y+1}, .., a_{XY}\rangle\rangle$$
$$\varepsilon_1\langle a_1\rangle \mapsto a_1$$

Let $f : \mathsf{D}_Y A \to B$, which uses its parameter Y times. Its extension, where the result B is used X times, $f^{\dagger}_{X,Y} : \mathsf{D}_{X\times Y} A \to \mathsf{D}_X B$, requires Y copies of the parameter A to compute each B, therefore $X \times Y$ copies of the parameter are needed. Delimited counit $(\varepsilon_1)_A : !_1 A \to A$ requires one use of the value. This delimited-comonad captures notions of data access and usage of the context.

A similar delimited comonad was suggested (but not defined) for a stream-based semantics of dataflow computation, where coeffects denote the number of past values accessed from the context [12].

Definition 3. *For a delimited comonad* $\mathsf{D} : \mathcal{I} \to [\mathcal{C}, \mathcal{C}]$, $CoK_{\mathcal{I}}(\mathcal{C})$ *is a* delimited coKleisli category *with* $|CoK_{\mathcal{I}}(\mathcal{C})| = |\mathcal{C}|$ *and morphisms* $CoK_{\mathcal{I}}(\mathcal{C})(A, B) = \bigcup_{X\in\mathcal{I}}(\mathsf{D}_X A, B)$*, where for all* $f : \mathsf{D}_Y A \to B, g : \mathsf{D}_X B \to C$ *composition is* $g \circ^{\mathsf{D}} f = g \circ f^{\dagger}_{X,Y} : \mathsf{D}_{X\bullet Y} A \to B$*, and for all objects* A *identity is* $id^{\mathsf{D}}_A = (\varepsilon_I)_A$*.*

Relating comonads and delimited comonads Delimited comonads collapse to regular comonads when $\mathcal{I}$ is a single-object monoidal category. Thus, all comonads are delimited comonads with a trivial coeffect.

However, not all delimited comonads are comonads, such as in the partiality delimited comonad. Further, for all $X \in |\mathcal{I}|$, D_X is not necessarily a comonad (*e.g.*, $D_\perp A = 1$ in Example 4). Whilst comonads exhibit shape preservation, delimited comonads may not be shape preserving, where for all $X, Y, f : \mathsf{D}_Y A \to B$ then $\mathsf{D}_X !_B \circ f^{\dagger}_{X,Y} \neq \mathsf{D}_{X\bullet Y} !_A$, since the parameter to D may determine shape.

Effect-delimited monads The effect-delimited monad structure arises as a *lax* monoidal functor $T : \mathcal{I} \to [\mathcal{C}, \mathcal{C}]$ for which coeffect-delimited comonads are exactly dual. The operations and laws of the lax monoidal structure are dual to delimited comonads, with [4]:

$$\eta_1 : I_{\mathcal{C}} \dot{\to} \mathsf{T}1 \qquad \mu_{F,G} : \mathsf{T}F \circ \mathsf{T}G \dot{\to} \mathsf{T}(F \bullet G)$$

As an informal example, the usual notion of *state monad* can be delimited with sets of the *read* and *write* effects where T can be refined with respect to the effect, *e.g.*, $\mathsf{T}\{\mathsf{read}\,\rho : \tau\}\,A = \tau \to A$ and $\mathsf{T}\{\mathsf{write}\,\rho : \tau\}\,A = A \times \tau$ (note: the latter is not itself a monad), or for mixed reading/writing $\mathsf{T}_X A = X \to (A \times X)$.

3 Delimited monoidal functors

Comonads and coeffect-delimited comonads give a semantics for sequential composition of contextual computations. However, a comonadic semantics for the λ-calculus requires the additional structure of a (lax) monoidal comonad which has a lax monoidal functor [14]. Previously, we showed the semantics of the coeffect calculus requires indexed forms of lax and also colax monoidal functors [12]. Here we generalise the derivation of delimited (co)monads to derive additional delimited monoidal structures (corresponding to these previous indexed notions).

Monoidal comonads provide a lax monoidal functor, with $\mathsf{m}_{A,B} : \mathsf{D}A \times \mathsf{D}B \to \mathsf{D}(A \times B)$ which is used to merge contexts in the semantics of abstraction [14]:

$$(\text{abs})\frac{[\![\Gamma, x : \sigma \vdash e : \tau]\!] = g : \mathsf{D}(\Gamma \times \sigma) \to \tau}{[\![\Gamma \vdash \lambda x.e : \sigma \to \tau]\!] = \Lambda(g \circ \mathsf{m}_{\Gamma,\sigma}) : \mathsf{D}\Gamma \to \tau^{\mathsf{D}\sigma}} \tag{2}$$

where $g \circ \mathsf{m}_{\Gamma,\sigma} : \mathsf{D}\Gamma \times \mathsf{D}\sigma \to \tau$ and Λ is the *currying* operation for exponents (*i.e.*, given $f : A \times B \to C$ then $\Lambda f : A \to C^B$). A monoidal comonad requires that the lax monoidal functor operations commute with the operations of a comonad (*e.g.*, $\mathsf{D}\mathsf{m}_{A,B} \circ \mathsf{m}_{A,B} \circ (\delta \times \delta) = \delta \circ \mathsf{m}_{A,B}$), which provides a semantics for the contextual λ-calculus with $\beta\eta$-equality (for details see [9, §3.4]).

For example, the product comonad has $\mathsf{m}_{A,B}((a, x), (b, y)) \mapsto ((a, b), x \otimes y)$ where $\otimes$ is associative. This gives a semantics where the context of a function body is the $\otimes$-combination of contexts at the declaration and call site.

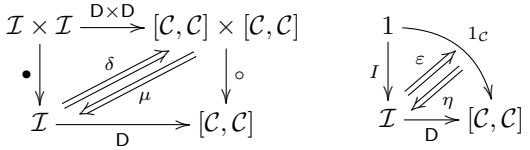
This monoidal structure is similarly required for a coeffect-delimited semantics, where an analogous, indexed monoidal comonad was previously proposed, with $\mathsf{m}^{X,Y}_{A,B} : \mathsf{D}_X A \times \mathsf{D}_Y B \to \mathsf{D}_{X \otimes Y}(A \times B)$ [12]. In the delimited setting, further colax monoidal structure is required to define *pairing* for context plumbing, with $\mathsf{n}^{X,Y}_{A,B} : \mathsf{D}_{X \oplus Y}(A \times B) \to \mathsf{D}_X A \times \mathsf{D}_Y B$, such that for $f : \mathsf{D}_X A \to B, g : \mathsf{D}_Y A \to C$:

$$\langle f, g \rangle^{\mathsf{D}} = (f \times g) \circ \mathsf{n}^{X,Y}_{A,A} \circ \mathsf{D}_{X \oplus Y} \Delta : \mathsf{D}_{X \oplus Y} A \to (B \times C) \tag{3}$$

The operations $\oplus$ and $\otimes$ combine coeffect information in $\mathsf{m}_{A,B}$ and $\mathsf{n}_{A,B}$, which may be distinct. Using different operations leads to a general delimited semantics, where each semantic operation may manipulate contexts differently.

Abstracting delimited structure The definition of delimited (co)monads essentially describes homomorphisms between structure in the category $\mathcal{I}$, which encodes program properties (*e.g.* (co)effects), and structure in the category $[\mathcal{C}, \mathcal{C}]$, which encodes impure computation. Delimited monads are a monoid homomorphism (on categories, *i.e.*, a lax monoidal functor) and delimited comonads are a monoid homomorphism in the reverse direction (*i.e.*, a colax monoidal functor).

The structure of these monoid homomorphisms is made clear in a diagram with 2-cells (morphisms between morphisms, written as double arrows $\Rightarrow$):



(4)

In the left square, following the top-right edge yields $X, Y \mapsto \mathsf{D}_X, \mathsf{D}_Y \mapsto \mathsf{D}_X\mathsf{D}_Y$ and the left-bottom $X, Y \mapsto X \bullet Y \mapsto \mathsf{D}_{X \bullet Y}$. Thus, δ is a natural transformation between $\mathsf{D}_{\_\bullet\_}$ and $\mathsf{D}_\_\mathsf{D}_\_ : \mathcal{I} \times \mathcal{I} \to [\mathcal{C}, \mathcal{C}]$, *i.e.*, $(\delta_{X,Y})_A : \mathsf{D}_{X \bullet Y} A \to \mathsf{D}_X \mathsf{D}_Y A$, providing delimited comultiplication. The dual natural transformation is then the delimited multiplication $\mu_{X,Y}$ of delimited monads.

In the right triangle, the left-bottom edge yields $1 \mapsto I \mapsto \mathsf{D}_I$ and the top/right edge $1 \mapsto 1_{\mathcal{C}}$. Thus, ε_I is the natural transformation between the functors D_I and $1_{\mathcal{C}} : 1 \to [\mathcal{C}, \mathcal{C}]$, *i.e.*, $(\varepsilon_I)_A : \mathsf{D}_I A \to A$, providing delimited counit. The dual is then the delimited unit η_I of delimited monads.

The same approach can be used to derive other delimited structures, by understanding what kind of homomorphism is needed between which structures. In the case of delimited lax monoidal functors, these are also monoid homomorphisms but involving products as monoids and composites of monoids.

Deriving delimited monoidal functors Consider $\mathsf{m}^{X,Y}_{A,B} : \mathsf{D}_X A \times \mathsf{D}_Y B \to \mathsf{D}_{X \otimes Y}(A \times B)$ in the semantics of abstraction. It takes a pair of two delimited contexts and returns a combined context, with coeffect information composed by $\otimes$ and free-variable contexts A and B concatenated by a product. Thus, products in the source and target of $\mathsf{m}^{X,Y}_{A,B}$ have different meanings: the former denotes the meta-language notion of pairing, the latter the object-language concept of context concatenation. We therefore derive $\mathsf{m}^{X,Y}_{A,B}$ as a homomorphism between a combined monoid of $\otimes$ and $\times$ (context concatenation) and $\times$ (context pairing).

Let $(\mathcal{I}, \otimes, E)$ be the monoid of coeffects describing context merging. Assume $\mathcal{C}$ has finite products and a terminal object, modelling free-variable context concatenation and empty contexts, providing the monoidal category $(\mathcal{C}, \times, 1)$.

In the target of $\mathsf{m}^{X,Y}_{A,B}$, the structure of free-variable contexts and the coeffects is coupled. We associate the two by taking the product category of the monoidal categories $(\mathcal{I}, \otimes, E)$ and $(\mathcal{C}, \times, 1)$, giving monoidal category $(\mathcal{I} \times \mathcal{C}, \otimes \bowtie \times, E \times 1)$, where $\bowtie : (\mathcal{I} \times \mathcal{C}) \times (\mathcal{I} \times \mathcal{C}) \to \mathcal{I} \times \mathcal{C}$ flips its arguments to apply the respective monoids, *i.e.*, $(\otimes \bowtie \times)\,((X, A), (Y, B)) \mapsto (X \otimes Y, A \times B)$.

We then *uncurry* D, written $\mathsf{D}^* : \mathcal{I} \times \mathcal{C} \to \mathcal{C}$, and define the notion of homomorphism between $(\mathcal{I} \times \mathcal{C}, \otimes \bowtie \times, E \times 1)$ and $(\mathcal{C}, \times, 1)$, which maps pairs of contexts to context concatenations, with associated coeffects. This is illustrated by the following 2-cell diagram which has the same shape/structure as the monoid homomorphisms that derive delimited (co)monads (diagram (4) above):

$$\begin{array}{ccc} (\mathcal{I} \times \mathcal{C}) \times (\mathcal{I} \times \mathcal{C}) & \xrightarrow{\mathsf{D}^* \times \mathsf{D}^*} & \mathcal{C} \times \mathcal{C} \\ {\scriptstyle \otimes \bowtie \times}\downarrow & \swarrow\!\!\swarrow {\scriptstyle \mathsf{m}^{X,Y}_{A,B}} & \downarrow{\scriptstyle \times} \\ \mathcal{I} \times \mathcal{C} & \xrightarrow[\mathsf{D}^*]{} & \mathcal{C} \end{array} \qquad \begin{array}{ccc} 1 & \xrightarrow{1} & \\ {\scriptstyle E \times 1}\downarrow & \swarrow {\scriptstyle \mathsf{m}^{E}_{1}} & \searrow \\ \mathcal{I} \times \mathcal{C} & \xrightarrow[\mathsf{D}^*]{} & \mathcal{C} \end{array} \tag{5}$$

where the top-right edge gives $(X, A), (Y, B) \mapsto (\mathsf{D}_X A, \mathsf{D}_Y B) \mapsto \mathsf{D}_X A \times \mathsf{D}_Y B$ and the left-bottom edge $(X, A), (Y, B) \mapsto (X \otimes Y, A \times B) \mapsto \mathsf{D}_{(X \otimes Y)}(A \times B)$.

The monoid homomorphism axioms give us delimited versions of the usual monoidal functor axioms of associativity for $\mathsf{m}^{X,Y}_{A,B}$ and unitality of the m^{E}_{1}. We say that D is a $(\mathcal{I}, \otimes, E)$-delimited lax monoidal functor (over products).

The dual homomorphism to the above, with a monoid $(\mathcal{I}, \oplus, 0)$, provides a delimited *co*lax monoidal functor, with $\mathsf{n}^{X,Y}_{A,B} : \mathsf{D}_{X\oplus Y}(A \times B) \to \mathsf{D}_X A \times \mathsf{D}_Y B$ and $\mathsf{n}^0_1 : \mathsf{D}_0 1 \to 1$. Note, we sometimes ignore the the unit operation, and thus deal in delimited lax *semi*-monoidal functors.

Delimited monoidal natural transformations A delimited (co)lax monoidal natural transformation is a delimited natural transformation (such as $\delta_{X,Y}$) which commutes with the operations of a delimited (co)lax monoidal functor.

For example, if $\delta_{X,Y}$ is a colax monoidal natural transformation with respect to the $(\mathcal{I}, \oplus, 0)$-delimited colax monoidal functor then:

$$\begin{array}{ccc} \mathsf{D}_{R\bullet X}A \times \mathsf{D}_{S\bullet Y}B & \xleftarrow{\quad\mathsf{n}_{R\bullet X, S\bullet Y}\quad} & \mathsf{D}_{(R\bullet X)\oplus(S\bullet Y)}(A\times B) \\ \downarrow{\scriptstyle \delta_{R,X}\times\delta_{S,Y}} & & \equiv \mathsf{D}_{(R\oplus S)\bullet(X\oplus Y)}(A\times B) \\ & & \downarrow{\scriptstyle \delta_{R\oplus S, X\oplus Y}} \\ \mathsf{D}_R\mathsf{D}_X A \times \mathsf{D}_S\mathsf{D}_Y B & \xleftarrow{\mathsf{n}_{R,S}} \mathsf{D}_{R\oplus S}(\mathsf{D}_X A \times \mathsf{D}_Y B) & \xleftarrow{\mathsf{D}\mathsf{n}_{X,Y}} \mathsf{D}_{R\oplus S}\mathsf{D}_{X\oplus Y}(A\times B) \end{array} \quad (6)$$

The top-right corner of this diagram necessarily equates $\mathsf{D}_{(R\bullet X)\oplus(S\bullet Y)}(A\times B) \equiv \mathsf{D}_{(R\oplus S)\bullet(X\oplus Y)}(A\times B)$ for the diagram to commute. This implies that $\bullet$ and $\oplus$ must mutually distribute, *i.e.* $(R \bullet X) \oplus (S \bullet Y) = (R \oplus S) \bullet (X \oplus Y)$.

Correspondence between identities on $\mathcal{I}$ and proofs The derivations here give a correspondence between algebraic identities on $\mathcal{I}$ and coherence conditions between morphisms in $\mathcal{C}$ (*e.g.*, between $X \bullet I = X$ and $\mathsf{D}_X\varepsilon_I \circ \delta_{X,I} = id_{\mathsf{D}_X}$). This assists in proving equality of denotations, *e.g.*, for $[\![\Gamma \,?\, R \vdash e_1 : \tau]\!] \stackrel{?}{=} [\![\Gamma \,?\, S \vdash e_2 : \tau]\!]$ a proof that $R = S$ suggests the coherence conditions to prove equality of the denotations. This is made more useful when each delimited structure (homomorphism) on D has a distinct algebraic structure on $\mathcal{I}$. This approach is employed in the next section to identify semantic conditions from equational theories.

Furthermore, a stronger result holds that if $f : \mathsf{D}_X A \to B$ and $g : \mathsf{D}_Y A \to B$ both uniquely factor through the delimited structures which give X and Y, then a proof of $X = Y$ maps directly to the proof that $f = g$, modulo naturality properties and coherence conditions arising from non-delimited structure on $\mathcal{C}$.

4 Coeffect-delimited semantics for a contextual λ-calculus

We previously defined a λ-calculus of contextual computations with a general coeffect system (parameterised by an abstract algebra), called the *coeffect calculus* [12]. The coeffect calculus has a denotational semantics in terms of an indexed comonad and indexed lax and colax monoidal functors, for which we have derived equivalent delimited structures in this paper. We redevelop the coeffect calculus semantics in more detail using these delimited structures. Each delimited structure uses a different abstract structure over $\mathcal{I}$, inducing the general notion of a *coeffect algebra*. We motivate its potential axioms arising from equational theories for the calculus. The semantics is type-and-coeffect directed so that the coeffect annotations match the denotations, *i.e.*:

$$[\![\Gamma \,?\, R \vdash e : \tau]\!] : \mathsf{D}_R[\![\Gamma]\!] \to [\![\tau]\!]$$

Uustalu and Vene previously defined a (non-delimited) comonadic semantics for a contextual λ-calculus following the Cartesian-closed semantics for the simply-typed λ-calculus [14]. We follow this approach over a delimited coKleisli category where standard components are written superscripted, *e.g.*, composition and pairing as $\circ^{\mathsf{D}}$ and $\langle -, - \rangle^{\mathsf{D}}$, which are then expanded to their definitions in $\mathcal{C}$.

Delimited semantics and coeffect system Figure 2 shows the delimited semantics. For (var), the standard semantics of projection from a context is used. For (abs), we use the $(\mathcal{I}, \otimes)$-delimited semi-monoidal functor derived in §3, where the semantics is derived from *currying*. This is analogous to the (non-delimited) comonadic semantics for abstraction using a lax semi-monoidal functor shown earlier (2). For (app), the standard semantics uses pairing and *uncurrying*. Recall from §3 that pairing for a delimited coKleisli category requires a delimited colax (semi)-monoidal functor (equation (3)). Uncurrying in the delimited coKleisli category also requires a delimited colax monoidal functor so that $\psi^{\mathsf{D}} f : \mathsf{D}_X A \to C^{(\mathsf{D}_Y B)}$ given $f : \mathsf{D}_{X \oplus Y}(A \times B) \to C$. Thus, $\mathsf{n}^{X,Y}_{A,B}$ serves to split contexts.

The semantics of (app) differs to the originally proposed semantics in the coeffect calculus [12, p.8, Fig.5]. The original semantics is a refinement, where:

$$[\![\Gamma ? R \oplus (S \bullet T) \vdash e_1 e_2 : \tau]\!] = \psi g_1 \circ (id \times g_2^{\dagger_{S,T}}) \circ \mathsf{n}_{R,S \bullet T} \circ \mathsf{D}\Delta : \mathsf{D}_{R \oplus (S \bullet T)} \Gamma \to \tau$$

The two are equivalent when $(A \oplus B) \bullet (C \oplus D) = (A \bullet C) \oplus (B \bullet D)$ and correspondingly when δ is a $(\mathcal{I}, \oplus)$-delimited colax (semi)-monoidal natural transformation. For maximum generality we continue with the non-refined semantics.

A notion of subcoeffecting arises naturally from morphisms in $\mathcal{I}$, where for all $f : X \to Y$ in $\mathcal{I}$, D must have a corresponding natural transformation $\mathsf{D}_f : \mathsf{D}_X A \to \mathsf{D}_Y A$. This is used in (sub), which requires an ordering $\leq$ on coeffects. One approach is to assume the existence of such an ordering where $X \leq Y$ iff $f : X \to Y$ in $\mathcal{I}$. Alternatively, this ordering can be induced from other operations over $\mathcal{I}$. For example, $(\mathcal{I}, \oplus)$ may be taken as an idempotent semigroup which induces a preorder $\leq$ where whenever $X \oplus Y = Y$ then $X \leq Y$.

$$\text{(var)}\frac{x : \tau \in \Gamma}{[\![\Gamma \ ? \ I \vdash x : \tau]\!] = \pi_i^{\mathsf{D}} = \pi_i \circ \varepsilon_I : \mathsf{D}_I \Gamma \to \tau}$$

$$\text{(abs)}\frac{[\![\Gamma, x : \sigma \, ? \, X \otimes Y \vdash e : \tau]\!] = g : \mathsf{D}_{X \otimes Y}(\Gamma \times \sigma) \to \tau}{[\![\Gamma \, ? \, X \vdash \lambda x.e : \sigma \xrightarrow{Y} \tau]\!] = \Lambda^{\mathsf{D}} g = \Lambda(g \circ \mathsf{m}^{X,Y}_{\Gamma,\sigma}) : \mathsf{D}_X \Gamma \to \tau^{(\mathsf{D}_Y \sigma)}}$$

$$\text{(app)}\frac{[\![\Gamma \, ? \, R \vdash e_1 : \sigma \xrightarrow{S} \tau]\!] = g_1 : \mathsf{D}_R \Gamma \to \tau^{(\mathsf{D}_S \sigma)} \quad [\![\Gamma \, ? \, T \vdash e_2 : \tau]\!] = g_2 : \mathsf{D}_T \Gamma \to \sigma}{\begin{array}{r}[\![\Gamma \, ? \, (R \oplus S) \bullet (I \oplus T) \vdash e_1 \, e_2 : \tau]\!] = \psi^{\mathsf{D}} g_1 \circ^{\mathsf{D}} \langle id^{\mathsf{D}}, g_2 \rangle^{\mathsf{D}} : \mathsf{D}_{(R \oplus S) \bullet (I \oplus T)} \Gamma \to \tau \\ = \psi g_1 \circ \mathsf{n}_{R,S} \circ ((\varepsilon_I \times g_2) \circ \mathsf{n}_{I,T} \circ \mathsf{D}\Delta)^{\dagger_{R \oplus S, I \oplus T}}\end{array}}$$

$$\text{(sub)}\frac{[\![\Gamma \, ? \, F \vdash e : \tau]\!] = g : \mathsf{D}_F \Gamma \to \tau \quad F' \leq F \Rightarrow f : F' \to F \in \mathcal{I}}{[\![\Gamma \, ? \, F' \vdash e : \tau]\!] = g \circ \mathsf{D}_f : \mathsf{D}_{F'} \Gamma \to \tau}$$

Fig. 2. Coeffect-delimited semantics for a contextual λ-calculus

Definition 4. *A* coeffect algebra *comprises a set* $\mathcal{I}$*, a monoid* $(\mathcal{I}, \bullet, I)$*, an idempotent semigroup* $(\mathcal{I}, \oplus)$ *inducing a preorder* $\leq$ *where* $X \oplus Y = Y \Rightarrow X \leq Y$*, and a semigroup* $(\mathcal{I}, \otimes)$*.*

Depending on what equational theory we have for a particular notion of context dependence, a coeffect algebra may have some additional axioms.

The semantics here induces a general coeffect system, shown in the judgments of Figure 2, parameterised by a coeffect algebra. This slightly generalises the previously proposed coeffect calculus, which had some built-in refinement [12].

Equational theory We consider standard $\beta\eta$-equality of the λ-calculus. For η-equality $(\lambda x.e\, x) \equiv e$, the type-and-coeffect derivation is:

$$\begin{array}{l}(\text{app})\\(\text{abs})\end{array} \frac{\dfrac{\Gamma, x : \sigma \,?\, R \vdash e : \sigma \xrightarrow{S} \tau \qquad \Gamma, x : \sigma \,?\, I \vdash x}{\Gamma, x : \sigma \,?\, (R \oplus S) \bullet (I \oplus I) \vdash e\, x : \tau}}{\Gamma \,?\, P \vdash \lambda x.e\, x : \sigma \xrightarrow{Q} \tau} \equiv \Gamma \,?\, R \vdash e : \sigma \xrightarrow{S} \tau$$

(where we elide weakening from $\Gamma \,?\, R \vdash e : \sigma \xrightarrow{S} \tau$ on the left-hand side). From idempotency of $\oplus$, $(R \oplus S) \bullet (I \oplus I) = (R \oplus S)$, thus η-equality holds if $R \otimes S = R \oplus S$, *i.e.*, implying that $\mathsf{n}^{R,S}_{A,B} \circ \mathsf{m}^{R,S}_{A,B} = id$ in the semantics.

For β-equality, we need first a meta-theorem on the coeffects of substitution:

Lemma 1. *Let* $\Gamma, x : \tau' \,?\, R \vdash e : \tau$ *and* $\Gamma \,?\, X \vdash e' : \tau'$*, then* $\Gamma \,?\, R \bullet (I \oplus X) \vdash e[x \mapsto e'] : \tau$*, given a* substituting coeffect algebra *(below).*

This is a more general version of that given previously [12, p.10], which had the coeffect $R \bullet (R \oplus X)$ instead. Its proof proceeds by induction over type-and-coeffect rules (Appendix A) eliciting conditions on $\mathcal{I}$ structures satisfied by the notion of a *substituting coeffect algebra*:

Definition 5. *A* substituting coeffect algebra *comprises a monoid* $(\mathcal{I}, \bullet, I)$*, an idempotent semi-group* $(\mathcal{I}, \oplus)$ *which induces a preorder* $\leq$ *where* $\forall X.I \leq X \vee X \leq I$*, a monoid* $(\mathcal{I}, \otimes, I)$*, and distributivity axioms:* $(A \bullet B) \oplus (C \bullet D) = (A \oplus C) \bullet (B \oplus D)$ *and* $(A \bullet B) \otimes (C \bullet D) = (A \otimes C) \bullet (B \otimes D)$*.*

This implies conditions on the underlying delimited structures (further to the $(\mathcal{I}, \bullet, I)$ coeffect-delimited comonad): D is a $(\mathcal{I}, \otimes, I)$-delimited lax monoidal functor, δ is $(\mathcal{I}, \oplus)$-delimited colax monoidal and $(\mathcal{I}, \otimes)$-delimited lax monoidal, and D is an idempotent delimited colax monoidal functor, *i.e.*, $\mathsf{n}^{X,X}_{A,A} \circ \mathsf{D}_X \Delta = \Delta$.

The property that I is the least or greatest element with respect to $\leq$ is needed for substitution over variables (see, Appendix A).

Given Lemma 1, β-equality then has type-and-coeffect derivation:

$$\begin{array}{l}(\text{abs})\\(\text{app})\end{array} \frac{\dfrac{\Gamma, x : \sigma \,?\, R \otimes S \vdash e : \tau}{\Gamma \,?\, R \vdash \lambda x.e : \sigma \xrightarrow{S} \tau} \quad \Gamma \,?\, T \vdash e' : \sigma}{\Gamma \,?\, (R \oplus S) \bullet (I \oplus T) \vdash (\lambda x.e) e' : \tau} \equiv \Gamma \,?\, (R \otimes S) \bullet (I \oplus T) \vdash e[x \mapsto e'] : \tau$$

This then holds when the coeffect algebra is a substituting coeffect algebra and $\oplus = \otimes$ (similarly to η-equivalence).

Example 6 (Implicit parameters). Recall the implicit parameters coeffect system of Figure 1. This has a coeffect algebra over the subsets of all possible variable-type pairs, written here T, with $(\mathcal{P}(T), \cup, \emptyset)$ monoid (for $\bullet$), $(\mathcal{P}(T), \cup)$ semigroup (for $\oplus$) and $(\mathcal{P}(T), \uplus, \emptyset)$ monoid (for $\otimes$). The induced ordering is that of subsetting. The additional delimited monoidal operations are straightforward:

$$\mathsf{n}^{X,Y}_{A,B}((a,b),x) \mapsto ((a, x-(X-Y)),(b, x-(Y-X)))$$
$$\mathsf{m}^{X,Y}_{A,B}((a,x),(b,y)) \mapsto ((a,b),(x \uplus y))$$

Since $\oplus \neq \otimes$ ($\uplus \neq \cup$) here then neither η or β-equality holds. However, a relaxed version of implicit parameters with possibly non-deterministic coeffect inference uses $\cup$ for all operations, providing full $\beta\eta$-equality.

Example 7 (Bounded reuse / dataflow). For bounded reuse/dataflow $\mathsf{D}_N A = A^N$, we define a coeffect algebra over $\mathbb{N}$, with $(\mathbb{N}, \times, 1)$ monoid (for $\bullet$), $(\mathbb{N}, max)$ semigroup (for $\oplus$) and $(\mathbb{N}, min)$ semigroup (for $\otimes$), with:

$$\mathsf{n}^{X,Y}_{A,B}\langle (a_1,b_1), \ldots (a_{X max Y}, b_{X max Y})\rangle \mapsto (\langle a_1, \ldots, a_X\rangle, \langle b_1, \ldots, b_Y\rangle)$$
$$\mathsf{m}^{X,Y}_{A,B}(\langle a_1, \ldots, a_X\rangle, \langle b_1, \ldots, b_Y\rangle) \mapsto \langle (a_1,b_1), \ldots, (a_{X min Y}, b_{X min Y})\rangle$$

This coeffect calculus does not have $\beta\eta$-equality. However, it does exhibit identity-like axioms on abstraction: $(\lambda x.x)e \equiv e$ and $(\lambda x.e)y = e[x \mapsto y]$ (essentially the (var) cases for substitution, Appendix A) which follow from the induced preorder from $(\mathbb{N}, max)$ and that 1 is the least element of this lattice.

5 Effect-delimited joinads

Whilst effect-delimited monads unify a denotational semantics with traditional effect systems (for sequential composition), many programs use other notions of composition, such as parallel composition and alternation (choice).

The notion of a *joinad* gives a model for parallelism and (possibly non-deterministic) choice in an effectful language [11]. Joinads comprise a monadic functor T and operations $par_{A,B} : \mathsf{T}A \times \mathsf{T}B \to \mathsf{T}(A \times B)$, $choice_A : \mathsf{T}A \times \mathsf{T}A \to \mathsf{T}A$, and $fail : 1 \to \mathsf{T}1$. This structure is a semantic counterpart to the work of Nielson and Nielson, who gave richer abstract algebraic structure to effect systems, separating the traditional lattices of effects into operations for sequential composition and alternation [8]. We unify joinads with similar richer effect annotations by generalising joinads to *effect-delimited joinads*.

Definition 6. *An effect-delimited joinad is an effect-delimited monad T, with effect monoid $(X, \bullet, I)$, and a $(X, |)$-delimited lax semi-monoidal functor, giving:*

$$par^{X,Y}_{A,B} : \mathsf{T}_X A \times \mathsf{T}_Y B \to \mathsf{T}_{X|Y}(A \times B)$$

and a lax monoidal functor on T between $(\mathcal{C}, \times, 1)$ and $(\mathcal{I}, +, 0)$ (the same construction that derives delimited (co)monads), giving:

$$choice^{X,Y}_A : \mathsf{T}_X A \times \mathsf{T}_Y A \to \mathsf{T}_{X+Y} A$$
$$fail^0_1 : 1 \to \mathsf{T}_0 1$$

Delimited joinads have additional equations capturing the original joinad laws:

- $(X, \bullet, I)$ and $(X, +, 0)$ form a semiring (where $+$ is the additive operation), where $0 \bullet X = 0 = X \bullet 0$ means sequentially composing with a failing computation is equivalent to a failing computation, and $X \bullet (Y + Z) = (X \bullet Y) + (X \bullet Z)$ means a sequentially composed computation can be distributed inside of a non-deterministic choice.
- $(X, \bullet, I)$ and $(X, |, I)$ forms a *pre*semiring (without absorption) (where $|$ is the additive operation), and $X \bullet (Y|Z) = (X \bullet Y)|(X \bullet Z)$ means a sequentially composed computation can be distributed inside of a parallel computation.

Thus, we can use delimited joinads extend a language and effect system with parallelism, choice, and failure.

6 Discussion

Annotated notions of (co)monads Wadler and Thiemann gave a syntactic correspondence between type-and-effect systems and a monadic semantics by annotating monadic type constructors with effect sets, T^F [15]. They gave soundness results between the effect system and operational semantics, and conjectured a coherent denotational semantics of effects and monads. One suggestion was to associate to each effect set F a different monad T^F. Katsumata's approach of effect-delimited monads, and the approach of this paper, unify (co)effect systems and (co)monads in the denotational semantics, where each T^σ is not necessarily a (co)monad (*i.e.*, T is not an indexed-family of structures).

Atkey's *parameterised monads* define a functor $\mathsf{T} : \mathcal{S}^{\mathsf{op}} \times \mathcal{S} \times \mathcal{C} \to \mathcal{C}$ where $\mathcal{S}$ is a category of *state descriptions* as objects and *state manipulations* as morphisms [1]. An object $\mathsf{T}(S_1, S_2, A)$ captures a computation that starts in state S_1 and ends in state S_2, yielding a value A. The operations of a parameterised monad are then $\eta^S_A : A \to \mathsf{T}(S, S, A)$ and $\mu^{S_1,S_2,S_3}_A : \mathsf{T}(S_1, S_2, (\mathsf{T}(S_2, S_3, A))) \to \mathsf{T}(S_1, S_3, A)$. Since categories are partial monoids, parameterised monads can be embedded into an effect-delimited monad for $((\mathcal{S} \times \mathcal{S}^{\mathsf{op}}) + \{\bot, I\}, \bullet', I)$ where

$$(S1, S2) \bullet' (S2', S3) = \begin{cases} (S1, S3) & S2 = S2' \\ \bot & \textit{otherwise} \end{cases}$$

and where I is a distinguished element representing a universal identity.

Embedding delimited monads into parameterised monads seems more difficult. Monoids can be encoded as single-object categories with elements of the monoid as morphisms. However, μ above is parameterised by objects, thus it cannot be parameterised directly by the monoid's elements. Parameterised monads have the advantage of allowing program logics (such as Hoare logic) to be encoded without resorting to partial binary operations. Further work is to unify the two approaches, generalising monads to be delimited by monoids or categories.

Tate introduced *productors* (capturing *producer effects*) to give a general annotated notion of effectful computation [13] into which can be fit other forms

of indexing, such as semi-lattice indexed families of monads and effect-delimited monads (see discussion in [4]). Tate suggested the dual notion of *consumer effects* (*consumptors*) for which coeffect-delimited comonads appear to be a special case. Further work is to dualise Tate's work to see if coeffects fit the approach.

The correspondence between comonads and S4 modal necessity $\Box$ is well known. Nanevski *et al.* introduced contextual modal type theory (CMTT) which extends an S4 term language with an indexed modality $[\Psi]\tau$ representing computations requiring a variable context Ψ [7]. Gabbay *et al.* give a denotational semantics to CMTT with a model of the modality approximately of the form $[v_1 : B_1, \ldots, v_n : B_n]A = A^{B_1 \times \ldots B_n}$ and suggest various comonad structures in the semantics [2, pp.33-35]. Further work is to see if CMTT admits a delimited comonad treatment or whether coeffects can be generalised to include CMTT.

Metalanguage for coeffects Previous work on monads and effects considered *metalanguages* with monadic type constructors, *i.e.*, first class monads in a language (*e.g.*, [6,15]). Our approach instead considered language semantics. Further work is to embed coeffect-delimited comonads into a metalanguage, which should be relatively straightforward given the semantic development here. We have already succeeded in embedding coeffect-delimited comonads in Haskell.

Structural notions of coeffect Example 5 defined a delimited comonad for Girard *et al.*'s bounded reuse connective. Recall the storage axiom that $!_{\bar{y}}\Gamma \vdash A \Rightarrow !_{x\bar{y}}\Gamma \vdash !_x A$ where $\bar{y}$ is a vector of equal size/structure to Γ implying that each variable in Γ has its own reuse bound. In the semantics of §4, the coeffect-delimited comonad $!_X$ instead ranges over the entire free-variable context, therefore approximating bounded reuse over all variables in the context. This is restrictive. Another restricted example is the partiality-delimited comonad, which approximates liveness over all variables in the context.

We previously suggested the notion of *structural coeffects* which extends the coeffect calculus to allow coeffect information for individual variables, rather than the entire context [12]. The development in this paper suggests a derivation of this system by relaxing $(\mathcal{I}, \otimes)$ and $(\mathcal{I}, \oplus)$ to *lax* monoids rather than *strict* monoids, and letting $\oplus = \otimes$ such that the $(\mathcal{I}, \otimes)$-delimited lax monoidal functor and $(\mathcal{I}, \oplus)$-delimited colax monoidal functor are isomorphisms. We can then modify the system to have explicit structural rules that manipulate the $\otimes$ structure, and strengthen the existing rules such that they do not admit contraction, exchange, and weakening. For example, (app) becomes:

$$(\text{app})\frac{\Gamma \,?\, R \vdash e_1 : \sigma \xrightarrow{S} \tau \qquad \Gamma' \,?\, T \vdash e_2 : \tau}{\Gamma, \Gamma' \,?\, R \otimes (S \bullet T) \vdash e_1\, e_2 : \tau}$$

Thus, $\otimes$ gives a coeffect annotation whose structure matches the structure of the free-variable context, giving per-variable coeffects. Initial research suggests that both systems can be accommodated in a general coeffect system (as described above), where the "flat" notion of coeffects (over a context) and "structural" notions (over variables) are both useful. This opens up coeffects to other applications such as secure information-flow properties.

Concluding remarks Effect and coeffect systems capture different important notions of computational impurity. The two differ most significantly in the (abs) rule, where abstraction is impure in its coeffects (sharing requirements between declaration and call site). We believe contextual notions of computation are important and that more work is needed to unify and model their semantics.

The general approach of deriving delimited structure via homomorphisms gives a useful technique for producing analyses coupled with denotational semantics, which we applied for the contextual λ-calculus and briefly for effectful parallelism, choice, and failure. We generalised our previously proposed coeffect calculus [12], provided a more detailed semantics motivating the structures and axioms, and made progress on extending this to structural coeffects.

Acknowledgments Thanks to Marcelo Fiore, Alan Mycroft and Tarmo Uustalu for discussion and help with categorical formulations, Murdoch Gabbay for suggesting the nominal discourse for static semantics of implicit parameters, Marco Gaboardi for suggesting the bounded-linear logic example and Shinya Katsumata for delimited monad discussion. This work is supported by CHESS and EPSRC.

References

1. R. Atkey. Parameterised notions of computation. *J. Funct. Program.*, 19(3-4):335–376, 2009.
2. M. J. Gabbay and A. Nanevski. Denotation of contextual modal type theory (CMTT): syntax and metaprogramming. *Journal of Applied Logic*, 2012.
3. J.-Y. Girard, A. Scedrov, and P. J. Scott. Bounded linear logic: a modular approach to polynomial-time computability. *Theoretical computer science*, 97(1):1–66, 1992.
4. S. Katsumata. Effect-delimited monad. In *POPL 2014 (To Appear)*, January 2014.
5. J. Lewis, J. Launchbury, E. Meijer, and M. Shields. Implicit parameters: Dynamic scoping with static types. In *Proceedings of the 27th ACM SIGPLAN-SIGACT symposium on Principles of programming languages*, page 118. ACM, 2000.
6. E. Moggi. Notions of computation and monads. *Inf. Comput.*, 93(1):55–92, 1991.
7. A. Nanevski, F. Pfenning, and B. Pientka. Contextual modal type theory. *ACM Trans. Comput. Logic*, 9(3):23:1–23:49, June 2008.
8. F. Nielson and H. Nielson. Type and effect systems. *Correct System Design*, pages 114–136, 1999.
9. D. Orchard. Programming contextual computations. PhD thesis, University of Cambridge, 2013. (To appear).
10. D. Orchard and A. Mycroft. A Notation for Comonads. In *IFL '12 post-proceedings*, volume 8241. Springer, September 2012.
11. T. Petricek, A. Mycroft, and D. Syme. Extending Monads with Pattern Matching. In *Proceedings of Haskell Symposium*, Haskell 2011, 2011.
12. T. Petricek, D. Orchard, and A. Mycroft. Coeffects: Unified Static Analysis of Context-Dependence. In *ICALP (2)*, pages 385–397, 2013.
13. R. Tate. The sequential semantics of producer effect systems. In *Proceedings of POPL 2013*, pages 15–26, 2013.
14. T. Uustalu and V. Vene. Comonadic Notions of Computation. *Electron. Notes Theor. Comput. Sci.*, 203(5):263–284, 2008.
15. P. Wadler and P. Thiemann. The marriage of effects and monads. *ACM Trans. Comput. Logic*, 4:1–32, January 2003.

Included for review process. To be turned into a technical report at a later date.

A Appendix: substitution lemma for contextual λ-calculus with coeffects

Lemma 2. *Let* $\Gamma.x : \tau' \,?\, R \vdash e : \tau$ *and* $\Gamma \,?\, X \vdash e' : \tau'$*, then* $\Gamma \,?\, R \bullet (I \oplus X) \vdash e[x \mapsto e'] : \tau$*, given a* substituting coeffect algebra*.*

Proof. By induction on type-and-coeffect rules.

– (var) $(\Gamma, x : \tau') \,?\, I \vdash v : \tau$ $\therefore$ substitution should yield a coeffect $I \bullet (I \oplus X) = I \oplus X$. By the definition of substitution there are two possible cases for $v[x \mapsto e']$:

1. $v = x$ $\therefore x[x \mapsto e'] = e'$ and $\tau = \tau'$, and therefore $\Gamma \,?\, X \vdash x[x \mapsto e'] : \tau'$. From the lemma's premise, I is either the lower- or upper-bound of $\leq$:
   - $\forall x.I \leq x$ (least) $\therefore I \oplus X = X$.
   - $\forall x.x \leq I$ (greatest) $\therefore I \oplus X = I$, then use (sub) so that $\Gamma \,?\, I \vdash e' : \tau'$.
2. $v \neq x$ $\therefore v[x \mapsto e'] = v$, $\therefore \Gamma \,?\, I \vdash v[x \mapsto e'] : \tau$.
   - $\forall x.I \leq x$ (least) $\therefore I \oplus X = X$, then use (sub) so that $\Gamma \,?\, X \vdash v : \tau$
   - $\forall x.x \leq I$ (greatest) $\therefore I \oplus X = I$.

– (app) Let $\Gamma, x : \tau' \,?\, R \vdash e_1 : \sigma \xrightarrow{S} \tau$ and $\Gamma, x : \tau' \,?\, T \vdash e_2 : \sigma$ then $\Gamma, x : \tau' \,?\, (R \oplus S) \bullet (I \oplus T) \vdash e_1\ e_2 : \tau$. Given that $(e_1\ e_2)[x \mapsto e'] = e_1[x \mapsto e']\, e_2[x \mapsto e']$, then inductive hypotheses on e_1 and e_2 gives the following derivation:

$$(app)\frac{\Gamma \,?\, R \bullet (I \oplus X) \vdash e_1[x \mapsto e'] : \sigma \xrightarrow{S} \tau \quad \Gamma \,?\, T \bullet (I \oplus X) \vdash e_2[x \mapsto e'] : \sigma}{\Gamma \,?\, ((R \bullet (I \oplus X)) \oplus S) \bullet (I \oplus (T \bullet (I \oplus X))) \vdash e_1[x \mapsto e']\, e_2[x \mapsto e'] : \tau} \quad (7)$$

By the lemma's statement, the result of substitution should have coeffect $\Gamma \,?\, ((R \oplus S) \bullet (I \oplus T)) \bullet (I \oplus X) \vdash (e_1\ e_2)[x \mapsto e'] : \tau$. The proof of equality between this coeffect and (7) relies on the monoidal properties of $\bullet$ (*i.e.*, that D is a delimited comonad), distributivity of $\bullet$ and $\oplus$ (*i.e.*, δ is colax monoidal), and idempotency of $\oplus$ (*i.e.*, D is an idempontent delimited colax monoidal functor):

$$\begin{array}{lll}
 & ((R \bullet (I \oplus X)) \oplus S) \bullet (I \oplus (T \bullet (I \oplus X))) & (7) \\
= & ((R \bullet (I \oplus X)) \bullet I) \oplus (S \bullet (T \bullet (I \oplus X))) & \{distrib\} \\
= & (R \bullet (I \oplus X)) \oplus (S \bullet (T \bullet (I \oplus X))) & \{unit\ \bullet\} \\
= & (R \bullet (I \oplus X)) \oplus ((S \bullet T) \bullet (I \oplus X))) & \{assoc\ \bullet\} \\
= & (R \oplus (S \bullet T)) \bullet ((I \oplus X) \oplus (I \oplus X))) & \{distrib\} \\
= & (R \oplus (S \bullet T)) \bullet (I \oplus X) & \{idem\} \\
= & (R \bullet I) \oplus ((S \bullet T) \bullet X) & \{distrib\} \\
(1) = & R \oplus ((S \bullet T) \bullet X) & \{idem\}
\end{array}$$

$$\begin{array}{lll}
 & ((R \oplus S) \bullet (I \oplus T)) \bullet (I \oplus X) & (goal) \\
= & ((R \bullet I) \oplus (S \bullet T)) \bullet (I \oplus X) & \{distrib\} \\
= & (R \oplus (S \bullet T)) \bullet (I \oplus X) & \{unit\ \bullet\} \\
= & (R \bullet I) \oplus ((S \bullet T) \bullet X) & \{distrib\} \\
= & R \oplus ((S \bullet T) \bullet X) & \{unit\ \bullet\} \\
= & (1) \quad \square &
\end{array}$$

– (abs) Let $\Gamma, x : \tau', v : \sigma \,?\, S \otimes T \vdash e : \tau$ then $\Gamma, x : \tau' \,?\, S \vdash \lambda v.e : \sigma \xrightarrow{T} \tau$. Given that $(\lambda v.e)[x \mapsto e'] = \lambda v.(e[x \mapsto e'])$ (assuming α-renaming such that $x \neq v$), the inductive hypothesis on e gives the derivation:

$$(abs)\frac{\Gamma, v : \sigma \,?\, (S \otimes T) \bullet (I \oplus X) \vdash e[x \mapsto e'] : \tau}{\Gamma \,?\, P \vdash (\lambda v.e[x \mapsto e']) : \sigma \xrightarrow{Q} \tau}[P \otimes Q = (S \otimes T) \bullet (I \oplus X)] \qquad (8)$$

By the lemma's statement, the result of substitution should have the coeffect $\Gamma \,?\, (S \bullet (I \oplus X)) \vdash (\lambda v.e)[x \mapsto e'] : \sigma \xrightarrow{T} \tau$. Therefore, for substitution to hold then $P = S \bullet (I \oplus X)$ and $Q = T$ and $(S \bullet (I \oplus X)) \otimes T = (S \otimes T) \bullet (I \oplus X)$:

$$\begin{array}{ll} & (S \bullet (I \oplus X)) \otimes T & (goal) \\ = & (S \bullet (I \oplus X)) \otimes (T \bullet I) & \{unit\ \bullet\} \\ = & (S \otimes T) \bullet ((I \oplus X) \otimes I) & \{distrib\} \\ = & (S \otimes T) \bullet (I \oplus X) & \{unit\ \otimes\} \\ = & (8) \quad \square & \end{array}$$

Thus, this holds if $\bullet$ and $\otimes$ distribute (δ is lax monoidal nat. trans. wrt $\mathsf{m}^{X,Y}_{A,B}$), and that I is the unit of $\otimes$ (*i.e.*, D is a $(\mathcal{I}, \otimes, I)$-delimited lax monoidal functor). *i.e.*, $\mathsf{m}^I_1 : 1 \to \mathsf{D}_I 1$ and $\mathsf{D}_X A \times 1 \xrightarrow{id \times \mathsf{m}^I_1} \mathsf{D}_X A \times \mathsf{D}_I 1 \xrightarrow{\mathsf{m}^{X,I}_{A,1}} \mathsf{D}_{X \otimes I}(A \times 1) \xrightarrow{\mathsf{D}\pi_2} \equiv id$.