Under consideration

Factorisation systems for logical relations and monadic lifting in type-and-effect system semantics

Ohad Kammar[1]

Department of Computer Science
University of Oxford
Oxford, England

Dylan McDermott[2]

Computer Laboratory
University of Cambridge
Cambridge, England

Abstract

Type-and-effect systems incorporate information about the computational effects, e.g., state mutation, probabilistic choice, or I/O, a program phrase may invoke alongside its return value. A semantics for type-and-effect systems involves a parameterised family of monads whose size is exponential in the number of effects. We derive such refined semantics from a single monad over a category, a choice of algebraic operations for this monad, and a suitable factorisation system over this category. We relate the derived semantics to the original semantics using fibrations for logical relations. Our proof uses a folklore technique for lifting monads with operations.

Keywords: computational effects, type-and-effect systems, monads, factorisation systems, fibrations, logical relations, denotational semantics

1 Introduction

Consider the following program phrase in an imperative-functional ML-like language:

```
1  let (triple:unit→int) = λ_:unit. 3*(get ℓ)
2  in ℓ := 1;
3     ℓ := triple() + triple()
```

The locally-defined function triple : unit→int triples the value read from memory location ℓ. The phrase then triples this value twice, and mutates the state to the sum of these two results. When optimising the program, we would like to cache the call to triple, and replace line 3 with a single memory access:

```
3     ℓ := let y = triple() in y + y
```

This transformation only preserves the semantics because the computational effects triple invokes are limited to reading. If we replace instead its definition on line 1 with a function that increments location ℓ' with each invocation of triple then the caching optimisation is no longer semantics preserving:

---

[1] Email: ohad.kammar@cs.ox.ac.uk

[2] Email: dylan.mcdermott@cl.cam.ac.uk

```
let (triple:unit→int) = λ_:unit. ℓ′ := (1 + get ℓ′); 3*(get ℓ)
```

Type-and-effect systems [20] refine types, such as triple : unit→int, to propagate the information about which computational effects code pieces may invoke, e.g., decorating function types with additional *effect annotations*:

$$\mathtt{triple} : \mathtt{unit} \xrightarrow{\varepsilon} \mathtt{int}$$

In *Gifford-style* systems, these annotations are finite sets of effect operations, such as $\varepsilon := \{\mathrm{get}, \mathrm{set}\}$. For example, for every proper subset $\varepsilon \subset \{\mathrm{get}, \mathrm{set}\}$, the caching transformation for every function $f : A \xrightarrow{\varepsilon} B$ is semantics preserving, while for $\varepsilon = \{\mathrm{get}, \mathrm{set}\}$ it is not.

Adequate denotational semantics is a natural technique for validating such equational transformations, and there is a long line of work validating type-and-effect-dependent transformations, starting with independent results by Tolmach [30], Wadler [31], and Benton et al. [3], and continuing to this day [2]. In their most general form, the semantics for an effect system consists of a *graded monad* [15], a compatible family of monad-like structures T_ε indexed by the effect annotations ε.

Here we make two contributions:

Contribution 1: avoiding structural combinatorial blow-up. To give the model structure for an arbitrary Gifford-style type-and-effect system with n operation symbols, one would need to give the structure of 2^n different monad-like structures, $n2^{n-1}$ monad-like-morphisms, and commute more than the same amount of diagrams to discharge the relevant proof obligations. To circumvent this blow-up, for example, Benton et al. give uniform bespoke definitions for each T_ε, e.g., as in [2]. To avoid an ad-hoc definition for each collection of effects, Katsumata [15] constructs graded monads for Gifford-style systems when the effects in the language are free. Here we give a general construction for Gifford-style systems whose effects are given by a set of Kleisli arrows for an arbitrary monad over a category with a factorisation system with appropriate closure properties, providing a uniform construction even when the effects of interest are not free.

Contribution 2: relationship to a base semantics. We also show that this construction gives sound and complete reasoning principles with respect to the original semantics under additional assumptions. As usual, such proofs involve constructing a logical relation. Here, we work fibrationally using Katsumata's notion of a *fibration for logical relations* [14]. We extend Hughes and Jacobs's characterisation of fibrations of factorisation systems [7] and characterise the factorisation systems that correspond to fibrations for logical relations. Finally, we also define generally a monadic lifting for an arbitrary monad along a fibration for logical relations that also lifts a given collection of Kleisli arrows. This construction utilises the bijection between algebraic operations and generic effects [25]. While Kammar [10] describes it in the special set-theoretic case, we believe this folklore monadic lifting methodology [3] should be known in its greater generality. We demonstrate that our results are applicable in several cases of interest.

These two contributions substantially generalise Kammar and Plotkin's previous domain-theoretic [11] and set-theoretic constructions [10]. Our factorisation system construction also strictly generalises the one in Kammar's thesis [10], which is limited to factorisation of enriched Lawvere theories [28] over a locally presentable category. The development here is also substantially simpler than Kammar's thesis. This simplification occurs in two levels. Kammar's previous development requires a combinatorial solution set condition argument using Bousfield's factorisation theorem [4], while our factorisation construction is structural and elementary. Second, our proofs are straightforward in comparison.

The rest of the paper is structured as follows. Section 2 presents our main factorisation construction. Section 3 uses this construction to give semantics for a type-and-effect system for Moggi's computational λ-calculus. Section 4 instruments a logical relations soundness and completeness proof from the factorisation construction. Section 5 surveys example applications to our construction. Section 6 concludes.

2 Factorising monads

To present our main construction, we first review the relevant category theoretic concepts and results.

2.1 Preliminaries and terminology

We assume familiarity with category theory, including categories $\mathcal{C}$, $\mathcal{D}$, functors $F, G : \mathcal{C} \to \mathcal{D}$, and natural transformations $\alpha, \beta : F \to G$, and related concepts as found in textbooks such as Mac Lane's [21].

2.1.1 Factorization systems

A factorisation system axiomatises the set-theoretic situation in which every function $f : A \to B$ can be factorised as $f = m \circ e$, i.e., a surjection $e : A \twoheadrightarrow f[A]$ onto the image of f, followed by the injection

[3] Alex K. Simpson, private communication, 2015.

$m : f[A] \rightarrowtail B$ of this image into f's codomain. In the general situation, we have two classes of morphisms $(\mathcal{E}, \mathcal{M})$ over a category $\mathcal{C}$, where $\mathcal{E}$-morphisms are thought of as epimorphisms and $\mathcal{M}$-morphisms are thought of as monomorphisms. We adopt the common convention to reserve the notation $e : A \twoheadrightarrow B$ for an $\mathcal{E}$-morphism and $m : B \rightarrowtail C$ for an $\mathcal{M}$-morphism when $\mathcal{E}$ and $\mathcal{M}$ are clear from the context, but emphasise that neither class needs to consist of epis or monos.

Definition 2.1 An *orthogonal factorisation system* on a category $\mathcal{C}$ is a pair $(\mathcal{E}, \mathcal{M})$ consisting of two classes of morphisms of $\mathcal{C}$ such that:

- Both $\mathcal{E}$ and $\mathcal{M}$ are closed under composition, and contain all isomorphisms.
- Every morphism $f : X \to Y$ in $\mathcal{C}$ factors into $f = m \circ e$ for some $m \in \mathcal{M}$ and $e \in \mathcal{E}$.
- The *diagonal fill-in* property is satisfied: for each commutative square as on the left, with $m \in \mathcal{M}$ and $e \in \mathcal{E}$ there is a unique morphism $h : X \to Y$ such that $h \circ m = f$ and $e \circ h = g$, as on the left below:

$$
\begin{tikzcd}
W \arrow[r, two heads, "e"] \arrow[d, "f"'] \arrow[dr, phantom, "="] & X \arrow[d, "g"] \\
Y \arrow[r, tail, "m"'] & Z
\end{tikzcd}
\implies
\begin{tikzcd}
W \arrow[r, two heads, "e"] \arrow[d, "f"'] \arrow[dr, phantom, "=" near start] & X \arrow[d, "g"] \arrow[dl, dashed, "h"] \arrow[dl, phantom, "=" near end] \\
Y \arrow[r, tail, "m"'] & Z
\end{tikzcd}
\qquad
\begin{tikzcd}
X \arrow[r, "f"] \arrow[d, "g_1"'] \arrow[dr, phantom, "="] & Y \arrow[d, "g_2"] \\
X' \arrow[r, "f'"'] & Y'
\end{tikzcd}
\implies
\begin{tikzcd}
X \arrow[r, two heads, "e"] \arrow[d, "g_1"'] \arrow[dr, phantom, "="] & A \arrow[r, tail, "m"] \arrow[d, dashed, "h"] \arrow[dr, phantom, "="] & Y \arrow[d, "g_2"] \\
X' \arrow[r, two heads, "e'"'] & A' \arrow[r, tail, "m'"'] & Y'
\end{tikzcd}
$$

Under the first two axioms, the diagonal fill-in axiom is equivalent to a form of functoriality in factorisation, as in the implication above on the right. In addition, it implies that factorisations of morphisms are unique up to a unique canonical iso, and so we talk about *the* factorisation of a morphism.

Example 2.2 The category **Set** has (surjection, injection) as a factorisation system, i.e., $\mathcal{E}$ is the class of surjective functions and $\mathcal{M}$ is the class of injective functions.

Example 2.3 [Meseguer [23]] Consider the category $\omega\mathbf{Cpo}$ of partial orders possessing all least upper bounds (lubs) of ω-indexed monotone sequences (ω-chains), i.e., ω-cpos, and monotone functions between them preserving these lubs, i.e., Scott-continuous functions. A *dense function* is a continuous function $e : X \twoheadrightarrow Y$ such that the smallest ω-chain-closed subset $U \subseteq Y$ with $e[X] \subseteq U$ is Y itself, i.e., a Scott-continuous function with a dense image. A *full function* is a continuous function $m : X \rightarrowtail Y$ such that $m\,x \leq m\,x'$ implies $x \leq x'$ for each $x \in X$. The category $\omega\mathbf{Cpo}$ has (dense, full) as a factorisation system. Every full function is necessarily injective, but there are non-surjective dense functions [19].

Example 2.4 Consider the functor category $[\mathcal{W}, \mathcal{C}]$, for a small category $\mathcal{W}$ and any category $\mathcal{C}$, and let $(\mathcal{E}, \mathcal{M})$ be a factorisation system on $\mathcal{C}$. Take $\mathcal{E}'$ (respectively $\mathcal{M}'$) as the class of natural transformations that are component-wise in $\mathcal{E}$ (respectively $\mathcal{M}$). Then $(\mathcal{E}', \mathcal{M}')$ is a factorisation system for $[\mathcal{W}, \mathcal{C}]$.

The left and right classes in a factorisation system have useful closure properties. For example, if $g \circ f$ and f are in $\mathcal{E}$, then so is g. For another example, view both classes as full subcategories of the arrow category $\mathcal{C}^{\to}$ whose objects are triples $f = (A_1^f, A_2^f, \underline{f})$ consisting of a morphism $\underline{f} : A_1^f \to A_2^f$, and whose morphisms $h : f \to g$ are pairs (h_1, h_2) consisting of morphisms $h_i : A_i^f \to A_i^g$ making the evident square commute. Then the left class is closed under colimits in the arrow category, and similarly the right class is closed under limits.

2.1.2 Monad structures and monads

The main feature of our factorisation construction is its modularity. First, factorisation takes place on a purely structural level, and we need no semantic properties such as the the monad laws. Second, factorisation takes place on a pay-as-you-go basis, factorising any additional data the morphism of interest preserves. To describe it explicitly, we first describe precisely the structures we will factorise.

A *monad structure* T on a category $\mathcal{C}$ consists of a triple $\left(\underline{T}, \mathrm{return}^T, \mu^T\right)$ where:

- the *functor part* $\underline{T}$ assigns to every $\mathcal{C}$-object A another $\mathcal{C}$-object $\underline{T}A$, and to every $\mathcal{C}$-morphism $f : A \to B$ another $\mathcal{C}$-morphism $\underline{T}f : \underline{T}A \to \underline{T}B$;
- the *unit* return^T assigns to every $\mathcal{C}$-object A a $\mathcal{C}$-morphism $\mathrm{return}^T_A : A \to \underline{T}A$; and
- the *multiplication* μ^T assigns to every $\mathcal{C}$-object A a $\mathcal{C}$-morphism $\mu^T_A : \underline{T}^2 A \to \underline{T}A$.

A *monad* is thus a monad structure T satisfying the well-known monad laws. When $\mathcal{C}$ has finite products, a *strong monad structure* is a monad structure T with an additional structure component:

- the *strength* str^T assigns to every pair of $\mathcal{C}$-objects A and B a $\mathcal{C}$-morphism $\mathrm{str}^T_{A,B} : A \times \underline{T}B \to \underline{T}(A \times B)$.

A *strong monad* is thus a strong monad structure satisfying the well-known laws. We similarly define *Kleisli triple structures* $T = \left(\underline{T}, \mathrm{return}^T, \gg\!=^T\right)$, demanding only an assignment $\underline{T}$ on object and a Kleisli extension operation $\gg\!=^{T}_{A,B} f : \underline{T}A \to \underline{T}B$ for every $f : A \to \underline{T}B$. Finally, when $\mathcal{C}$ is cartesian closed, we define a *strong Kleisli triple structure* $T = (\underline{T}, \mathrm{return}^{T}, \gg\!=^{T})$ analogously, replacing $\gg\!=$ with an assignment of a morphism $\gg\!=^{T}_{A,B} : \underline{T}A \times \underline{T}B^{A} \to \underline{T}B$ for every pair of $\mathcal{C}$-objects A and B.

Morphisms of such structures $m : S \to T$ assign to every $\mathcal{C}$-object A a morphism $m_A : \underline{S}A \to \underline{T}A$ that preserve the structure, i.e., satisfy the same conditions a (strong) monad morphism should. Such morphisms provide super-categories for the categories of (strong) monads and (strong) Kleisli triples. The usual isomorphisms between the familiar sub-categories fail to extend to isomorphisms between the structural super-categories without the presence of the monad laws.

An *algebra structure* $A = (\underline{A}, \mathrm{alg}_A)$ for a monad structure T over $\mathcal{C}$ consists of:

- the *carrier* $\underline{A}$, a $\mathcal{C}$-object; and
- the *algebra map* alg_A, a $\mathcal{C}$-morphism $\mathrm{alg}_A : \underline{T}\underline{A} \to \underline{A}$.

When T is a monad, an *algebra* is an algebra structure satisfying the well-known algebra properties. Similarly, when T is a Kleisli triple structure, an *algebra structure* $A = (\underline{A}, \gg\!=_A)$ replaces the algebra map with:

- the *extension operator* $\gg\!=_A$ which assigns to every morphism $f : X \to \underline{A}$ a morphism $\gg\!= f : \underline{T}X \to \underline{A}$.

When T is a Kleisli triple, an *algebra* is an algebra structure satisfying [22], for every $f : X \to \underline{A}$, and every $g : X \to \underline{T}Y$, $h : Y \to \underline{A}$:

$$\begin{tikzcd} X \arrow[r, "\mathrm{return}"] \arrow[dr, "f"'] & \underline{T}X \arrow[d, "\gg= f"] \arrow[dl, phantom, "="] \\ & \underline{A} \end{tikzcd} \qquad \begin{tikzcd} \underline{T}X \arrow[r, "\gg= g"] \arrow[dr, "\gg=^{A}((\gg=^{A} h)\circ g)"'] & \underline{T}Y \arrow[d, "\gg= h"] \arrow[dl, phantom, "="] \\ & \underline{A} \end{tikzcd}$$

Similarly, we define an algebra structure for a strong Kleisli triple structure by replacing the extension operator with an internal extension operator $\gg\!= : \underline{T}X \times \underline{A}^{X} \to \underline{A}$, and algebras for a strong Kleisli triple internalise the two equations above.

Let T be a monad over a category $\mathcal{C}$. Recall that a *Kleisli arrow* is a morphism $f : A \to \underline{T}B$. When $\mathcal{C}$ is cartesian closed and T is strong, an *algebraic operation* [25] $\alpha : A \to B$ for T assigns to every $\mathcal{C}$-object X a $\mathcal{C}$-morphism $\alpha_X : (\underline{T}X)^{B} \to (\underline{T}X)^{A}$, natural in X, and respecting the multiplication/extension and the strength. Plotkin and Power [25] establish a bijection between Kleisli arrows $f : A \to \underline{T}B$ and algebraic operations $\alpha : A \to B$ given by:

$$\mathrm{uncurry}\,\alpha_X : A \times (\underline{T}X)^{B} \xrightarrow{f \times \mathrm{id}} \underline{T}B \times (\underline{T}X)^{B} \xrightarrow{\gg=} \underline{T}X$$

Let $F : \mathcal{C} \to \mathcal{C}$ be a functor with a tensorial strength str^{F} over a category with finite products. The category $F\text{-}\mathbf{Mnd}^{\mathcal{C}}$ of F-monads on $\mathcal{C}$ has as objects (T, β) where T is a strong monad and $\beta : F \circ T \to T$ is a natural transformation making the square on the left commute:

$$\begin{tikzcd} X \times F(T\,Y) \arrow[r, "\mathrm{str}^{F}"] \arrow[d, "\mathrm{id}\times\beta"'] & F(X \times T\,Y) \arrow[r, "F\,\mathrm{str}^{T}"] \arrow[d, phantom, "="] & F(T(X \times Y)) \arrow[d, "\beta"] \\ X \times T\,Y \arrow[rr, "\mathrm{str}^{T}"'] & & T(X \times Y) \end{tikzcd} \qquad \begin{tikzcd} F(T\,X) \arrow[r, "F\,m"] \arrow[d, "\beta"'] \arrow[dr, phantom, "="] & F(T'\,X) \arrow[d, "\beta'"] \\ T\,X \arrow[r, "m"'] & T'\,X \end{tikzcd}$$

A morphisms $m : (T, \beta) \to (T', \beta')$ consists of a strong monad morphism $m : T \to T'$ making the square on above right commute.

An *effect signature* ε in a category $\mathcal{C}$ consists of a set $\underline{\varepsilon}$ and an $\underline{\varepsilon}$-indexed family of pairs of $\mathcal{C}$-objects. We write $(\mathrm{op} : X \to Y) \in \varepsilon$ when $\mathrm{op} \in \underline{\varepsilon}$ and (X, Y) is the op-th component in ε. We write $\varepsilon \subseteq \varepsilon'$ when $\underline{\varepsilon} \subseteq \underline{\varepsilon}'$ and both agree component-wise.

For every effect signature ε we define the functor $F_\varepsilon : \mathcal{C} \to \mathcal{C}$ by $F_\varepsilon := \sum_{(\mathrm{op}:X\to Y)\in\varepsilon} X \times (-)^{Y}$. Every $F_\varepsilon\text{-}\mathbf{Mnd}^{\mathcal{C}}$-object (T, β) induces an algebraic operation α_{op} for each operation in $(\mathrm{op} : X \to Y) \in \varepsilon$, which in turn induces a Kleisli arrow $(T, \beta)\,[\![\mathrm{op}]\!] : X \to \underline{T}Y$. This process extends to an isomorphism between $F_\varepsilon\text{-}\mathbf{Mnd}^{\mathcal{C}}$ and the category whose objects are *ε-monads* on $\mathcal{C}$, i.e., pairs $(T, [\![-]\!])$ consisting of a strong monad T together with a morphism $[\![\mathrm{op}]\!] : X \to T\,Y$ for each $(\mathrm{op} : X \to Y) \in \varepsilon$. Its morphisms $m : (T, [\![-]\!]) \to (T', [\![-]\!])$ are strong monad morphisms $m : T \to T'$ such that, for all $(\mathrm{op} : X \to Y) \in \varepsilon$, we have $m \circ [\![\mathrm{op}]\!] = [\![\mathrm{op}]\!]$.

We recall Kelly's [17,18] transfinite construction of the free F-monad when $\mathcal{C}$ has κ-directed colimits and F is *κ-ranked*, i.e., preserves these colimits, for some regular cardinal κ. Define an ordinal-indexed sequence of functors $S_\alpha : \mathcal{C} \to \mathcal{C}$ by transfinite induction on α as follows:

$$S_0 \coloneqq \mathrm{Id} \qquad S_{\alpha+1} \coloneqq \mathrm{Id} + F \circ S_\alpha \qquad S_\lambda \coloneqq \operatorname*{colim}_{\alpha<\lambda} S_\alpha \qquad (\lambda \text{ a limit ordinal})$$

Each colimit is directed: the diagram includes morphisms $S_\alpha \to S'_\alpha$ for $\alpha \leq \alpha'$; each morphism is defined by transfinite recursion. The free monad for F is then given by $S_F \coloneqq \mathrm{colim}_{\alpha<\kappa} S_\alpha$. If F is also strong then S_F is the initial object of $F\text{-}\mathbf{Mnd}^{\mathcal{C}}$.

2.2 The factorisation theorems

Let $(\mathcal{E}, \mathcal{M})$ be a factorisation system for a category $\mathcal{C}$, and let S be monad structures on $\mathcal{C}$. We say that $(\mathcal{E}, \mathcal{M})$ is *closed under* S when, for every $e : A \twoheadrightarrow B$ in $\mathcal{E}$, we have $\underline{S}e : \underline{S}A \twoheadrightarrow \underline{S}B$ in $\mathcal{E}$. We also say that S is *compatible* with $(\mathcal{E}, \mathcal{M})$. In that case, we can factorise every monad structure morphism $m : S \to T$ through a monad structure $m[S]$ as a composition of monad structure morphisms $m : S \twoheadrightarrow m[S] \rightarrowtail T$ by choosing a factorisation for each m_X, setting for each $f : X \to Y$, and Z:

$$\begin{tikzcd}
\underline{S}X \arrow[rr, "m_X"] \arrow[dr, two heads, "m^{\mathbf{e}}_X"'] & \arrow[d, phantom, ":="] & \underline{T}X \\
& \underline{m[S]}X \arrow[ur, tail, "m^{\mathbf{m}}_x"'] &
\end{tikzcd}
\qquad
\begin{tikzcd}
\underline{S}X \arrow[r, two heads, "m^{\mathbf{e}}_X"] \arrow[d, "\underline{S}f"'] & \underline{m[S]}X \arrow[d, tail, "m^{\mathbf{m}}_X"] \arrow[dl, dashed, "\underline{m[S]f}"'] \\
\underline{S}Y \arrow[d, two heads, "m^{\mathbf{e}}_Y"'] & \underline{T}X \arrow[d, "\underline{T}f"] \\
\underline{m[S]}Y \arrow[r, tail, "m^{\mathbf{m}}_Y"'] & \underline{T}Y
\end{tikzcd}
\qquad
\begin{tikzcd}
\underline{Z} \arrow[rr, "\mathrm{return}^{m[S]}_Z"] \arrow[dr, "\mathrm{return}^S_Z"'] & \arrow[d, phantom, ":="] & \underline{m[S]}Z \\
& \underline{S}Z \arrow[ur, two heads, "m^{\mathbf{e}}_Z"'] &
\end{tikzcd}$$

$$\begin{tikzcd}
\underline{S}^2Z \arrow[r, two heads, "\underline{S}m^{\mathbf{e}}_Z"] \arrow[d, "\mu^S_Z"'] & \underline{S}\underline{m[S]}Z \arrow[r, two heads, "m^{\mathbf{e}}_{m[S]Z}"] & \underline{m[S]}^2Z \arrow[d, tail, "m^{\mathbf{m}}_Z"] \arrow[dddll, dashed, "\mu^{m[S]}_Z"'] \\
\underline{S}Z \arrow[ddd, two heads, "m^{\mathbf{e}}_Z"'] & & \underline{m[S]}\underline{T}Z \arrow[d, "\underline{m[S]}m^{\mathbf{m}}_Z"] \\
& & \underline{T}^2Z \arrow[d, "\mu^T_Z"] \\
\underline{m[S]}Z \arrow[rr, tail, "m^{\mathbf{m}}_Z"'] & & \underline{T}Z
\end{tikzcd}$$

This definition makes $m^{\mathbf{e}} : S \twoheadrightarrow m[S]$ a monad structure morphism with components in $\mathcal{E}$, and $m^{\mathbf{m}} : m[S] \rightarrowtail T$ a monad structure morphism with components in $\mathcal{M}$. Using the factorisation system closure properties, $(\mathcal{E}, \mathcal{M})$ is also closed under $m[S]$. Moreover, we have a (component-wise $\mathcal{E}$, component-wise $\mathcal{M}$) factorisation system of the category of $(\mathcal{E}, \mathcal{M})$-compatible monad structures and monad structure morphisms. Every algebra structure A for $m[S]$ induces an algebra structure S by setting:

$$\mathrm{alg}^T_A : \underline{S}A \overset{m^{\mathbf{e}}_A}{\twoheadrightarrow} \underline{m[S]}\underline{A} \xrightarrow{\mathrm{alg}_A} \underline{A}$$

When $\mathcal{C}$ has finite products, we say that a factorisation system $(\mathcal{E}, \mathcal{M})$ is *closed under products* when, for every $e_1, e_2 \in E$, we also have that $e_1 \times e_2 \in \mathcal{E}$. We can then factorise a strong monad structure morphism $m : S \to T$ by setting the strength for $m[S]$ as on the left:

$$\begin{tikzcd}
X \times \underline{S}Y \arrow[r, two heads, "\mathrm{id} \times m^{\mathbf{e}}_Y"] \arrow[d, "\mathrm{str}^S_{X,Y}"'] & X \times \underline{m[S]}Y \arrow[d, tail, "\mathrm{id} \times m^{\mathbf{m}}_Y"] \arrow[dl, dashed, "\mathrm{str}^{m[S]}_{X,Y}"'] \\
\underline{S}(X \times Y) \arrow[d, two heads, "m^{\mathbf{e}}_{X\times Y}"'] & X \times \underline{T}Y \arrow[d, "\mathrm{str}^T_{X,Y}"] \\
\underline{m[S]}(X \times Y) \arrow[r, tail, "m^{\mathbf{m}}_{X\times Y}"'] & T(X \times Y)
\end{tikzcd}
\qquad
\begin{tikzcd}
\underline{S}X \times \left(\underline{m[S]}Y\right)^X \arrow[r, two heads, "m^{\mathbf{e}}_X \times \mathrm{id}"] \arrow[dd, ">\!\!>\!\!=^S"'] & \underline{m[S]}X \times \left(\underline{m[S]}Y\right)^X \arrow[d, "m^{\mathbf{m}}_X \times (m^{\mathbf{m}}_Y)^X"] \arrow[ddl, dashed, ">\!\!>\!\!=^{m[S]}"'] \\
& \underline{T}X \times (\underline{T}Y)^X \arrow[d, ">\!\!>\!\!=^T"] \\
\underline{m[S]}Y \arrow[r, tail, "m^{\mathbf{m}}_Y"'] & \underline{T}Y
\end{tikzcd}$$

We also include the factorisation construction for strong Kleisli triples in a cartesian closed category, above on the right. This construction uses the fact that algebra structures for $m[S]$ induce algebra structures for S.

Theorem 2.5 (Factorisation) *Let $\mathcal{C}$ be a category, $(\mathcal{E}, \mathcal{M})$ a factorisation system, S and T be monads over $\mathcal{C}$, and $m : S \to T$ a monad morphism.*

- *If $(\mathcal{E}, \mathcal{M})$ is closed under S then $m[S]$ is a monad, and so $m^{\mathbf{e}}$ and $m^{\mathbf{m}}$ are monad morphisms. As a consequence, every algebra for $m[S]$ induces an algebra for S.*
- *If, moreover, $(\mathcal{E}, \mathcal{M})$ is closed under products, S, T are strong monads, and so m is a strong monad morphism, then $m[S]$ is a strong monad and $m^{\mathbf{e}}$, $m^{\mathbf{m}}$ are strong monad morphisms.*
- *When, moreover, $\mathcal{C}$ is cartesian closed, the constructions for Kleisli triples and strong monads coincide.*

The proof, commuting several diagrams, uses the diagonal fill-in property by substituting definitions.

We can transfer additional structure from S to $m[S]$. Post-composing with $m^{\mathbf{e}}$ transfers to $m[S]$ any Kleisli arrow for S. Let $F : \mathcal{C} \to \mathcal{C}$ be a strong functor and assume $(\mathcal{E}, \mathcal{M})$ is closed under F. If (S, β) and (T, β') are objects of $F\text{-}\mathbf{Mnd}^{\mathcal{C}}$ and m is a $F\text{-}\mathbf{Mnd}^{\mathcal{C}}$-morphism we equip $m[S]$ with a $F\text{-}\mathbf{Mnd}^{\mathcal{C}}$-object structure $(m[S], m[\beta])$ by setting as below on the left. We then have that $m^{\mathbf{e}}$ and $m^{\mathbf{m}}$ are $F\text{-}\mathbf{Mnd}^{\mathcal{C}}$-morphisms.

$$\begin{tikzcd}
F\underline{S}X \arrow[r, two heads, "Fm^{\mathbf{m}}_X"] \arrow[d, "\beta_X"'] & F\underline{m[S]}X \arrow[d, "Fm^{\mathbf{m}}_X"] \arrow[ddl, dashed, "m[\beta]"'] \\
\underline{S}X \arrow[d, two heads, "m^{\mathbf{e}}_X"'] \arrow[r, phantom, "="] & F\underline{T}X \arrow[d, "\beta'_X"] \\
\underline{m[S]}X \arrow[r, tail, "m^{\mathbf{m}}_X"'] & \underline{T}X
\end{tikzcd}
\qquad
\begin{tikzcd}
S \arrow[r, "m"] \arrow[d, "f_1"'] \arrow[dr, phantom, "="] & T \arrow[d, "f_2"] \\
S' \arrow[r, "m'"'] & T'
\end{tikzcd}
\implies
\begin{tikzcd}
S \arrow[r, two heads] \arrow[d, "f_1"'] \arrow[dr, phantom, "="] & m[S] \arrow[r, tail] \arrow[d, dashed, "m[f]"] \arrow[dr, phantom, "="] & T \arrow[d, "f_2"] \\
S' \arrow[r, two heads] & m[S'] \arrow[r, tail] & T'
\end{tikzcd}$$

Using the diagonal fill-in property, we can *functorially* factorise commuting squares of monad structure morphisms, i.e., morphisms $f = (f_1, f_2)$ between monad structure morphisms, as above on the right.

Theorem 2.6 (Functoriality) *Let $(\mathcal{E}, \mathcal{M})$ be a factorisation system for a category $\mathcal{C}$, and let $f : (S, T, m) \to (S', T', m')$ be a commuting square of monad structure morphisms. If $(\mathcal{E}, \mathcal{M})$ is closed under S and S', then $m[f] : m[S] \to m[S']$ is a monad structure morphism that preserves all of the above structure that f preserves:*

- *if $(\mathcal{E}, \mathcal{M})$ is closed under products and f is strong, then so is $m[f]$; and*
- *if moreover f is an F-monad structure morphism, then $m[f]$ is an F-monad structure morphism.*

So far, we have worked with an arbitrary factorisation system $(\mathcal{E}, \mathcal{M})$. When it is an *epi-mono* factorisation system, i.e., a pair $(\mathcal{E}, \mathcal{M})$ in which $\mathcal{M}$ consists of monos, then Theorem 2.5 holds under the weaker assumption that T is a monad, while S need only be a monad structure. To prove it, instead of appealing to the diagonal fill-in property, use the cancellation property of monos.

2.3 Free monads

To apply the Factorisation Theorem 2.5, we need to choose a suitable monad S and monad morphism m. When giving semantics to type-and-effect systems, we take S to be the free monad for the functor F_ε from the end of §2.1.2. Here we give a sufficient condition for F_ε, or more generally, any functor F, to be compatible with the factorisation system.

Lemma 2.7 *Let $\mathcal{C}$ be a category with κ-directed colimits, κ a regular cardinal, $F : \mathcal{C} \to \mathcal{C}$ be a κ-ranked functor, and $(\mathcal{E}, \mathcal{M})$ a factorisation system over $\mathcal{C}$. If F is compatible with $(\mathcal{E}, \mathcal{M})$, then the free F-monad S_F is compatible with $(\mathcal{E}, \mathcal{M})$.*

To apply the last lemma to the signature functor F_ε, we want to show that F_ε preserves κ-directed colimits for some κ, and that $\mathcal{E}$ is closed under F_ε. For colimit preservation, the following lemma covers our examples.

Lemma 2.8 *Let ε be an effect signature in a locally presentable cartesian closed category $\mathcal{C}$. Then the functor F_ε preserves κ-directed colimits for some regular cardinal κ.*

However, some F_ε may be incompatible with some factorisation systems:

Example 2.9 Consider the (dense, full) factorisation system on $\omega\mathbf{Cpo}$. Exponentials $(-)^Y$ preserve dense maps iff Y is a countable discrete ω-cpo. For a simple illustration, take the discrete natural numbers $\mathbb{N}$ and the ordinal $\omega + 1$. Take $Y := \omega + 1$, and consider the inclusion $e : \mathbb{N} \to \omega + 1$, which is a dense map. Every monotone function $f : \omega + 1 \to \mathbb{N}$ is constant, and so the ω-chain-closure of $e^Y[\mathbb{N}^Y]$ contains only constant functions. Therefore, the identity function $x := \mathrm{id} \in (\omega + 1)^Y$ is not in this closure, hence e^Y isn't dense.

3 Type-and-effect systems

We consider a variant of Moggi's [24] computational λ-calculus, λ_{c}, and its refinement with a Gifford-style type-and-effect system. The denotational semantics for such a system is standard, and we focus on the specific model structure given by the Factorisation Theorem 2.5.

3.1 Syntax

The syntax of λ_{c} are parametrised by three sets: a set $\mathcal{B}$ of *base types* ranged over by b; a set $\underline{\Sigma}$ of *operations* ranged over by op; and a set $\underline{\mathcal{K}}$ of *constants* ranged over by c. We also have the metavariable x range over

$$\frac{(c : A) \in \mathcal{K}}{\Gamma \vdash_{\emptyset} c : A} \quad \frac{\Gamma \vdash_{\varepsilon} M : A \qquad (\mathrm{op} : A \to B) \in \Sigma}{\Gamma \vdash_{\varepsilon \cup \{\mathrm{op}\}} \mathrm{op}\, M : B} \quad \frac{\Gamma \vdash_{\varepsilon} M : A \qquad \varepsilon \subseteq \varepsilon'}{\Gamma \vdash_{\varepsilon'} M : A} \quad \frac{(x : A) \in \Gamma}{\Gamma \vdash_{\emptyset} x : A} \quad \frac{}{\Gamma \vdash_{\emptyset} () : A}$$

$$\frac{\Gamma \vdash_{\varepsilon} M : A \qquad \Gamma \vdash_{\varepsilon'} N : B}{\Gamma \vdash_{\varepsilon \cup \varepsilon'} (M, N) : A \times B} \quad \frac{\Gamma \vdash_{\varepsilon} M : A \times B}{\Gamma \vdash_{\varepsilon} \mathsf{fst}\, M : A} \quad \frac{\Gamma \vdash_{\varepsilon} M : A \times B}{\Gamma \vdash_{\varepsilon} \mathsf{snd}\, M : B} \quad \frac{\Gamma \vdash_{\varepsilon} M : 0}{\Gamma \vdash_{\varepsilon} \mathsf{elim}_0\, M : A}$$

$$\frac{\Gamma \vdash_{\varepsilon} M : A}{\Gamma \vdash_{\varepsilon} \mathsf{inl}\, M : A + B} \quad \frac{\Gamma \vdash_{\varepsilon} M : B}{\Gamma \vdash_{\varepsilon} \mathsf{inr}\, M : A + B} \quad \frac{\Gamma \vdash_{\varepsilon} M : A_1 + A_2 \qquad \Gamma, x : A_1 \vdash_{\varepsilon'} N_1 : B \qquad \Gamma, y : A_2 \vdash_{\varepsilon'} N_2 : B}{\Gamma \vdash_{\varepsilon \cup \varepsilon'} \mathsf{match}\ M\ \mathsf{with}\ \{\mathsf{inl}\, x.\, N_1, \mathsf{inr}\, y.\, N_2\} : B}$$

$$\frac{\Gamma, x : A \vdash_{\varepsilon} M : B}{\Gamma \vdash_{\emptyset} \lambda x.\, M : A \xrightarrow{\varepsilon} B} \quad \frac{\Gamma \vdash_{\varepsilon} M : A \xrightarrow{\varepsilon''} B \qquad \Gamma \vdash_{\varepsilon'} N : A}{\Gamma \vdash_{\varepsilon \cup \varepsilon' \cup \varepsilon''} M\, N : B}$$

Fig. 1. λ_c type-and-effect system

some set of variables and ε ranges over finite subsets of $\underline{\Sigma}$. The syntax of types A, B (base types, products and sums, and function types), ground types G, and terms M of the λ_c-calculus is given as follows:

$$A, B ::= b \mid 1 \mid A \times B \mid 0 \mid A + B \mid A \xrightarrow{\varepsilon} B \qquad G ::= b \mid 1 \mid G_1 \times G_2 \mid 0 \mid G_1 + G_2$$

$$\begin{aligned} M, N ::= {} & c \mid \mathrm{op}\, M \mid x \mid () \mid (M, N) \mid \mathsf{fst}\, M \mid \mathsf{snd}\, M \mid \mathsf{elim}_0\, M \\ & \mid \mathsf{inl}\, M \mid \mathsf{inr}\, M \mid \mathsf{match}\ M\ \mathsf{with}\ \{\mathsf{inl}\, x.\, N_1, \mathsf{inr}\, y.\, N_2\} \\ & \mid \lambda x.\, M \mid M\, N \end{aligned}$$

The main difference to Moggi's calculus is that we include a specified set of constructs $\mathrm{op}\, M$ for causing effects. The other constructs are standard: built-in constants, unit value, products with projections, empty type elimination construct, sum injections and pattern matching, and function abstraction and application.

To define λ_c's type system, we need some typing information for effect operations and the constants. Formally, a λ_c *signature* is a triple $(\mathcal{B}, \Sigma, \mathcal{K})$ consisting of: a set $\mathcal{B}$ of base types; a family of pairs of ground type Σ indexed by a set of operations $\underline{\Sigma}$; and a family of types $\mathcal{K}$ indexed by a set $\underline{\mathcal{K}}$. We write $c : A$ when the type A is the c-component of $\mathcal{K}$, and $\mathrm{op} : G \to G'$ when (G, G') is the op-component of Σ.

Given a λ_c signature we define two type systems. The type-and-effect system consists of a typing judgment $\Gamma \vdash_{\varepsilon} M : A$ given inductively by the rules in Figure 1. Such judgements assert that in *typing context* Γ, a finitely supported partial function from variable names to types, the term M has type A and uses only the operations in $\varepsilon \subseteq \underline{\Sigma}$. The rules are standard for such systems.

We recover the usual type system for λ_c by erasing the effect annotations ε from the type syntax and from Figure 1. In detail, for each type A there is an erased type $\underline{A}$, and similarly for contexts Γ. The unrefined typing judgments $\underline{\Gamma} \vdash M : \underline{A}$ are generated by the rules of Figure 1 without annotations. This judgment places no constraints on the operations that M can use. We have that if $\Gamma \vdash_{\varepsilon} M : A$ then $\underline{\Gamma} \vdash M : \underline{A}$.

3.2 Semantics

Fix a λ_c signature $(\mathcal{B}, \Sigma, \mathcal{K})$. An *unrefined* λ_c *model*, consists of: a bicartesian closed category $\mathcal{C}$; an object $[\![b]\!] \in \mathcal{C}$ for each $b \in \mathcal{B}$; a Σ-monad T on $\mathcal{C}$; a Kleisli arrow $[\![\mathrm{op}]\!] : [\![G]\!] \to T\,[\![G']\!]$ for every $\mathrm{op} : G \to G'$ in Σ; and a morphism $[\![c]\!] : 1 \to [\![\underline{A}]\!]$ for each constant $(c : A) \in \mathcal{K}$. Unrefined models interpret the unrefined judgments $\underline{\Gamma} \vdash M : \underline{A}$, with types and contexts denoting $\mathcal{C}$-objects $[\![\underline{B}]\!]$ and $[\![\underline{\Gamma}]\!]$, and judgements denoting Kleisli arrows $[\![\underline{\Gamma} \vdash M : \underline{A}]\!] : [\![\Gamma]\!] \to T\,[\![\underline{A}]\!]$.

To interpret type-and-effect judgements in their greatest generality, one replaces the monad with a *graded monad* [15]. Here, as we restrict to Gifford-style systems, we consider a simpler structure. A *refined* λ_c *model* consists of: a bicartesian closed category $\mathcal{C}$; an object $[\![b]\!] \in \mathcal{C}$ for each $b \in \mathcal{B}$; these data allow us to interpret effect annotations ε as effect signatures $[\![\varepsilon]\!]$, and so we further require a functorial assignment T_{-}, to each $\varepsilon \subseteq \Sigma$, of an $[\![\varepsilon]\!]$-monad T_{ε} on $\mathcal{C}$, and to each inclusion $\varepsilon \subseteq \varepsilon'$ an $[\![\varepsilon]\!]$-monad morphism $T_{\varepsilon} \to T_{\varepsilon'}$; and a morphism $[\![c]\!] : 1 \to [\![A]\!]$ for each constant $(c : A) \in \mathcal{K}$. We interpret the refined judgement $\Gamma \vdash_{\varepsilon} M : A$ by a morphism $[\![\Gamma]\!] \to T_{\varepsilon}\,[\![A]\!]$ along the same lines of the unrefined semantics.

The main difference between the two model structures is the functorial assignment T_{-}, which requires additional structure over the unrefined model structure that is exponential in the number of operations. We can derive it in the following way and under the following assumptions, in addition to the unrefined model structure. First, we assume that, for each $\varepsilon \subseteq \Sigma$, we have the free ε-monad S_{ε}. Second, we assume a factorisation system $(\mathcal{E}, \mathcal{M})$ that is closed under products and each S_{ε}. By Lemmata 2.7 and 2.8 these two assumptions hold in any locally presentable cartesian closed category in which $\mathcal{E}$ is closed under exponentiation by the interpretation of base types. Third, we assume a $[\![\Sigma]\!]$-monad T. For every $\varepsilon \subseteq \Sigma$, by initiality of the free ε-monad, we have a unique monad morphism $m_\varepsilon : S_\varepsilon \to T$. Applying the Factorisation Theorem 2.5 to this monad morphism, we set $T_\varepsilon := m_\varepsilon[S_\varepsilon]$. Applying the functorial action of $m_\varepsilon[-]$ to the (unique) ε-monad morphism $S_{\varepsilon\subseteq\varepsilon'} : S_\varepsilon \to S_{\varepsilon'}$, we set $T_{\varepsilon\subseteq\varepsilon'} := m_\varepsilon[S_{\varepsilon\subseteq\varepsilon'}]$. Finally, we assume a refined interpretation of the built-in constants compatible with this structure.

3.3 Example reasoning

We demonstrate the model construction on a small set-theoretic example. Let $\mathbb{L}$ be a finite set of global memory location names. For our λ_c signature, we take: $\mathcal{B} := \{\mathrm{loc}, \mathrm{int}\}$, $\Sigma := \{\mathrm{get} : \mathrm{loc} \to \mathrm{int}, \mathrm{set} : \mathrm{loc} \times \mathrm{int} \to 1\}$, and

$$\mathcal{K} := \{+ : \mathrm{int} \times \mathrm{int} \xrightarrow{\emptyset} \mathrm{int}\} \cup \{\ell : \mathrm{loc} | \ell \in \mathbb{L}\} \cup \{a : \mathrm{int} | a \in \mathbb{Z}\}$$

For the unrefined model structure, we interpret: $[\![\mathrm{loc}]\!] := \mathbb{L}$ and $[\![\mathrm{int}]\!] := \mathbb{Z}$. For our monad, we set $\mathbb{S} := \mathbb{Z}^{\mathbb{L}}$ and take T to be the $\mathbb{S}$-state monad, $TX := (\mathbb{S} \times X)^{\mathbb{S}}$, with the usual interpretation for get and set. We interpret locations and integers as themselves, and $+$ as addition without side effects.

For the refined model, we take the (surjection, injection) factorisation system on **Set**. We can calculate that $T_{\{\mathrm{set}\}}X = (\mathbb{1} + \mathbb{Z})^{\mathbb{L}} \times X$ is the writer monad for the following *overwriting monoid* $\left((\mathbb{1} + \mathbb{Z})^{\mathbb{L}}, \mathbf{1}, *\right)$:

$$\mathbf{1} := (\iota_1 \star)_{\ell \in \mathbb{L}} \qquad \left((a_\ell)_{\ell \in \mathbb{L}} * (b_\ell)_{\ell \in \mathbb{L}}\right)_{\ell'} = \begin{cases} b_{\ell'} & b_{\ell'} \neq \iota_1 \star \\ a_{\ell'} & \text{otherwise} \end{cases}$$

I.e., an injected unit value at location ℓ represents no state change, while an injected integer a represents an update of that location to a. To see why, first note that the free $\{\mathrm{set}\}$-monad is the smallest set satisfying $S_{\{\mathrm{set}\}}X \cong X + \mathbb{L} \times \mathbb{Z} \times S_{\{\mathrm{set}\}}X$. The unique $\{\mathrm{set}\}$-monad morphism $m_{\{\mathrm{set}\}} : S_{\{\mathrm{set}\}} \to T$ satisfies:

$$m_{\{\mathrm{set}\}}(\iota_1 x) := \lambda s.\,(s, x) \qquad m_{\{\mathrm{set}\}}(\iota_2\,(\ell, a, r)) := \lambda s.\,\left(s[\ell \mapsto a], m_{\{\mathrm{set}\}}(r)\right)$$

Factorising it, and using the finiteness of $\mathbb{L}$, we get the surjection:

$$m^{\mathbf{e}}_{\{\mathrm{set}\}}(\iota_1 x) \mapsto (\iota_1 \star, x) \quad m^{\mathbf{e}}_{\{\mathrm{set}\}}(\iota_2\,(\ell, a, r)) \mapsto (((\iota_1 \star)_{\ell' \in \mathbb{L}}\,[\ell \mapsto \iota_2 a] * (-)) \times \mathrm{id})\left(m_{\{\mathrm{set}\}}(r)\right)$$

We then interpret $+$ as addition, as $T_\emptyset$ is the identity monad. We can then validate the example from the introduction, i.e. in the refined semantics $[\![M + M]\!] = [\![(\lambda x.x + x)M]\!]$ for every $\Gamma \vdash_{\{\mathrm{set}\}} M : \mathrm{int}$.

4 Monadic lifting

To prove that the refined factorisation semantics matches the unrefined semantics we use a suitable notion of logical relation. In this section we define a notion of *factorisation system for logical relations*, and show that these systems induce a suitable logical relation. This notion combines Hughes and Jacobs's [7] characterisation of fibrations arising from factorisation systems with Katsumata's [14] fibrations for logical relations.

4.1 Preliminaries

First we review some standard properties of fibrations, see Jacobs [8] for a systematic development of fibred category theory in type theory and logic. Instead of considering general fibrations, we will only consider the simpler case of faithful fibrations.

Let $p : \mathcal{D} \to \mathcal{C}$ be a faithful functor. For all $\mathcal{D}$-objects X, Y, we write $f : X \dot{\to} Y$ when $f : p\,X \to p\,Y$ in $\mathcal{C}$ and there is some (necessarily unique) $\dot{f} : X \to Y$ such that $p\,\dot{f} = f$. In this case we say that f *lifts* to $\dot{f}$. If $f : X \dot{\to} Y$ then $\dot{f}$ is *Cartesian* when, for all objects $Z \in \mathcal{D}$ and $g : p\,Z \to p\,X$ with $f \circ g : Z \dot{\to} X$ we have $g : Z \dot{\to} X$. The functor p is a *fibration* when, for every object Y in $\mathcal{D}$ and morphism $f : I \to p\,Y$ in $\mathcal{C}$ there is an object X such that $p\,X = I$ and $f : X \dot{\to} Y$ is Cartesian.

If $p : \mathcal{D} \to \mathcal{C}$ is a faithful fibration, we view objects $X \in \mathcal{D}$ as predicates over X, and morphisms $\dot{f} : X \to Y$ as truth-preserving maps. If $f : p\,X \to p\,Y$ then $f : X \dot{\to} Y$ means f is truth-preserving, and $\dot{f}$ is a witness to this preservation. Faithfulness implies that $\dot{f}$ is unique, so constructing such witnesses amounts to checking a property, instead of providing structure. Cartesianness of $\dot{f}$ intuitively means that X is true on as many elements of $p\,X$ as possible, with the constraint that f is truth-preserving.

For every $I \in \mathcal{C}$, the *fibre* $\mathcal{D}_I$ is the category consisting of objects $X \in \mathcal{D}$ such that $p\,X = I$ and morphisms $f : X \to Y$ in $\mathcal{D}$ such that $p\,f = \mathrm{id}_I$. We write $X \leq Y$ when there is a (necessarily unique) morphism from X to Y in $\mathcal{D}_I$, and $X \equiv Y$ when $X \leq Y$ and $Y \leq X$.

For each $f : I \to J$ in $\mathcal{C}$ there is an inverse image functor $f^* : \mathcal{D}_J \to \mathcal{D}_I$ that sends an object X to an object Y such that $f : X \dot{\to} Y$ is Cartesian. Such Y is unique up to $\equiv$: for any Y' with the same property we have $Y \equiv Y'$. We will also postulate that f^* has a left adjoint $f_* : \mathcal{D}_I \to \mathcal{D}_J$, the *direct image functor*. When f_* exists, we call p a *bifibration*.

For fibrations to give us logical relations, we also require both categories to be bicartesian closed, and require p to preserve the bi-cartesian closed structure. For example, products in $\mathcal{D}$ allow us to form logical relations over a product, and preservation of products implies that this relation has the usual property of logical relations. We will also want to form *conjunctions/intersections* of logical relations; these are given by products in fibres.

Katsumata combines all of these requirements into a single notion. A *fibration for logical relations* [14] over a bicartesian closed category $\mathcal{C}$ is a faithful fibration $p : \mathcal{D} \to \mathcal{C}$ such that:

- p is a bifibration: each inverse image functor f^* has a left adjoint f_*;
- $\mathcal{D}$ is bicartesian closed, and p strictly preserves the bicartesian closed structure; and
- each fibre $\mathcal{D}_I$ has all small products, denoted $\bigwedge$.

Our only deviation from Katsumata's definition is to require fibres to be pre-orders instead of partial orders, due to our use of non-*strict* factorisation systems.

Recall also the change-of-base construction which allows us to construct new fibrations for logical relations from existing ones:

Lemma 4.1 (Katsumata [14, Proposition 6]) *Let $p : \mathcal{D} \to \mathcal{C}$ be a fibration for logical relations, and let $F : \mathcal{C}' \to \mathcal{C}$ be a product-preserving functor. The projection from the pullback $F^*\,p$ of p along F is a fibration for logical relations on $\mathcal{C}'$.*

$$\begin{tikzcd}
F^*\,\mathcal{D} \arrow[r] \arrow[d, "F^*\,p"'] \arrow[dr, phantom, "\lrcorner", very near start] & \mathcal{D} \arrow[d, "p"] \\
\mathcal{C}' \arrow[r, "F"'] & \mathcal{C}
\end{tikzcd}$$

When we choose $F := (\times) : \mathcal{C} \times \mathcal{C} \to \mathcal{C}$, we call $F^*\,\mathcal{D}$ the category of binary logical p-relations over $\mathcal{C}$.

4.2 Fibrations from factorisation systems

Let $(\mathcal{E}, \mathcal{M})$ be a factorisation system on $\mathcal{C}$. The codomain functor $\mathrm{cod} : \mathcal{M} \to \mathcal{C}$ sends an $\mathcal{M}$-morphism $m : X \rightarrowtail Y$ to its codomain Y, recalling that we view $\mathcal{M}$ as a full subcategory of the arrow category $\mathcal{C}^{\to}$, so that objects are $\mathcal{M}$-monos and morphisms are commutative squares. Cartesian morphisms for cod are exactly pullback squares. Given an $\mathcal{M}$-morphism $m : X' \rightarrowtail Y'$ and a morphism $f : Y \to Y'$, we construct the Cartesian morphism required in the definition by taking the pullback of m along f:

$$\begin{tikzcd}
X \arrow[r] \arrow[d, tail, "f^*\,m"'] \arrow[dr, phantom, "\lrcorner", very near start] & X' \arrow[d, tail, "m"] \\
Y \arrow[r, "f"'] & Y'
\end{tikzcd}$$

$f^*\,m$ is necessarily in $\mathcal{M}$ due to the diagonal fill-in property. Hence if $\mathcal{C}$ has all pullbacks then cod is a fibration. If this is the case then cod is also a bifibration: the left adjoint f_* takes an $\mathcal{M}$-morphism m to the $\mathcal{M}$-morphism in the factorisation of $f \circ m$.

Example 4.2 Consider the (surjection,injection) factorisation for **Set**. Every injection $m : X \rightarrowtail Y$ is equal to the composition of an inclusion i and an isomorphism. In this case, we have $m \equiv i$. This fact rephrases that an injection is, up to isomorphism in the fibre, a subset $X \subseteq Y$. The direct image functor f_* of a function $f : Y \to Y'$ maps this subset to $\{f\,x \mid x \in X\} \subseteq Y'$. The inverse image functor f^* maps a subset $X' \subseteq Y'$ to $\{x \mid f\,x \in X'\} \subseteq Y$.

Example 4.3 Similarly for the (dense, full) factorisation for ω**Cpo**, the full functions are the chain-closed subsets. Inverse images are the usual inverse images, but direct images are now the ω-chain-closure of the direct image.

We extend the work of Hughes and Jacobs [7], who give a correspondence between certain factorisation systems (on categories with pullbacks) and fibrations with additional properties. We restrict this correspondence to fibrations for logical relations.

Definition 4.4 [cf. [7]] Let $\mathcal{C}$ be bicartesian closed. A factorisation system $(\mathcal{E}, \mathcal{M})$ over $\mathcal{C}$ is a *factorisation system for logical relations* when:

- *$\mathcal{C}$ has all pullbacks of $\mathcal{M}$-morphisms;*
- *every morphism in $\mathcal{M}$ is a monomorphism (i.e. $m \circ f = m \circ g \Rightarrow f = g$);*
- *for every $Y \in \mathcal{C}$ the fibre $\mathcal{M}_Y$ has small products;*
- *$\mathcal{M}$ is closed under binary coproducts; and*
- *$\mathcal{E}$ is closed under binary products.*

The monomorphism requirement implies that cod is faithful. The closure of $\mathcal{M}$ under coproducts implies that $\mathcal{M}$ is bicartesian (it automatically has initial and terminal objects and products). The closure of $\mathcal{E}$ under binary products implies that for $m' : X' \rightarrowtail Y'$ the canonical morphism $X \mathrel{\dot{\Rightarrow}} m' : X \Rightarrow X' \rightarrowtail X \Rightarrow Y'$ is an $\mathcal{M}$-morphism, and hence that $\mathcal{M}$ has exponentials, which are given by the following pullback:

$$\begin{tikzcd}
Z \arrow[r] \arrow[d, tail, "m \Rightarrow m'"'] \arrow[dr, phantom, "\lrcorner", very near start] & X \Rightarrow X' \arrow[d, tail, "X \Rightarrow m'"] \\
Y \Rightarrow Y' \arrow[r, "m \Rightarrow Y'"'] & X \Rightarrow Y'
\end{tikzcd}$$

Lemma 4.5 *Let $(\mathcal{E}, \mathcal{M})$ be a factorisation system over a bicartesian closed category $\mathcal{C}$. The codomain functor* $\mathrm{cod} : \mathcal{M} \to \mathcal{C}$ *is a fibration for logical relations iff $(\mathcal{E}, \mathcal{M})$ is a factorisation system for logical relations.*

This lemma also has a converse: if a fibration for logical relations is a *factorisation fibration* [7, Definition 3.1] then the induced factorisation system is a factorisation system for logical relations.

Example 4.6 The factorisation systems (surjection, injection) for **Set** and (dense, full) for ω**Cpo** are factorisation systems for logical relations. If $(\mathcal{E}, \mathcal{M})$ is a factorisation system for logical relations on $\mathcal{C}$, then (component-wise $\mathcal{E}$, component-wise $\mathcal{M}$) is a factorisation system for logical relations on $[\mathcal{W}, \mathcal{C}]$.

4.3 Folklore lifting for algebraic operations

Since our semantics uses monads, we also need to lift monads to the category of logical relations. Let $p : \mathcal{D} \to \mathcal{C}$ be a faithful fibration, ε is an effect signature in $\mathcal{D}$, and T be a $p\,\varepsilon$-monad on $\mathcal{C}$, where $p\,\varepsilon$ is the effect signature with operations $\mathrm{op} : p\,X \to p\,Y$ for $(\mathrm{op} : X \to Y) \in \varepsilon$. A *lifting* of T to $\mathcal{D}$ is an ε-monad on $\mathcal{D}$ such that:

- for each $X \in \mathcal{D}$ we have $p\,(\dot{T}\,X) = T\,(p\,X)$;
- for each $f : X \to Y$ we have $p\,(\dot{T}\,f) = T\,(p\,f)$;
- the unit lifts: $p\,(\mathrm{return}^{\dot{T}}) = \mathrm{return}^{T}$;
- the multiplication lifts: $p\,(\mu^{\dot{T}}) = \mu^{T}$;
- the strength lifts: $p\,(\mathrm{str}^{\dot{T}}) = \mathrm{str}^{T}$; and
- each $\mathrm{op} \in \varepsilon$ lifts: $p\,(\dot{\alpha}_{\mathrm{op}}) = \alpha_{\mathrm{op}}$.

Only the object action of $\dot{T}$ is a required structure, the other requirements are properties we need to check.

As each logical relations proof involving monads involves a lifting, these occur in abundance, and usually in an ad-hoc fashion. Two general lifting techniques are $\top\top$-lifting [13] and the codensity lifting [16]. The particular lifting we use is the *free lifting*, which is the ε-monad that is initial amongst all ε-liftings. The construction of this lifting is folklore, and is described for binary relations over **Set** in Kammar's thesis [10]. We describe it for the general case of a fibration for logical relations here.

Let $p : \mathcal{D} \to \mathcal{C}$ be a fibration for logical relations with *essentially small fibres*, i.e. each fibre has a representing set of objects up to $\equiv$. For each object $X \in \mathcal{D}$ define $\mathcal{R}X$ as the set of all X' in the representing set of $\mathcal{D}_{T\,(p\,X)}$ such that:

- The unit respects X': $\eta : X \mathrel{\dot{\to}} X'$.
- For each $(\mathrm{op} : A \to B) \in \varepsilon$ the algebraic operation α_{op} respects X': $\alpha_{\mathrm{op}} : B \mathrel{\dot{\Rightarrow}} X' \mathrel{\dot{\to}} A \mathrel{\dot{\Rightarrow}} X'$, where $\dot{\Rightarrow}$ denotes exponentials in $\mathcal{D}$.

This definition makes essential use of the bijection between algebraic operations and Kleisli arrows, as the former localises the closure condition to X' alone. The elements of $\mathcal{R}X$ can be thought of as candidates for $\dot{T}\,X$. We define the free lifting of T to $\mathcal{D}$ on objects by: $\dot{T}\,X \coloneqq \bigwedge \mathcal{R}X$, i.e., $\dot{T}\,X$ is the least element of $\mathcal{R}\,X$ with respect to the order $\leq$ in the fibre. This definition extends uniquely to an ε-monad $\dot{T}$ on $\mathcal{D}$.

Theorem 4.7 *$\dot{T}$ is a lifting of T to $\mathcal{D}$, and is initial: for all liftings $\dot{T}'$, the identity lifts to a (necessarily unique) ε-monad morphism $\dot{T} \to \dot{T}'$.*

4.4 Completeness

We now return to the language λ_c and relate the refined semantics we construct with the unrefined semantics. Suppose that the factorisation system we used to construct the refined semantics is a factorisation system for logical relations that is *well-powered*: each object has a representing set of $\mathcal{M}$-morphisms into it, and let $p : \mathbf{LogRel} \to \mathcal{C} \times \mathcal{C}$ be the fibration for logical relations constructed from the codomain fibration $\mathrm{cod} : \mathcal{M} \to \mathcal{C}$, as in Lemma 4.1. Explicitly, an object of $\mathbf{LogRel}$ is a triple (X, Y, m) where $m : Z \rightarrowtail X \times Y$ (for some Z) is an $\mathcal{M}$-mono. The *diagonal* relations are the objects (X, X, δ_X), where $\delta_X = \langle \mathrm{id}, \mathrm{id} \rangle : X \rightarrowtail X \times X$. We further assume that all diagonal relations exist, i.e., the diagonals δ_X are in $\mathcal{M}$. Well-poweredness of the factorisation system implies cod has essentially small fibres.

Example 4.8 The factorisation systems (surjection, injection) over **Set** and (dense, full) over ω**Cpo** are well-powered and have all diagonals. For every factorisation system $(\mathcal{E}, \mathcal{M})$ for $\mathcal{C}$ and every small category $\mathcal{W}$, the factorisation (component-wise $\mathcal{E}$, component-wise $\mathcal{M}$) is well-powered if $(\mathcal{E}, \mathcal{M})$ is well-powered, and has diagonals if $(\mathcal{E}, \mathcal{M})$ has diagonals.

Example 4.9 Over **Set**, the factorisation system (iso, any) is not well-powered, and the factorisation system (any, iso) does not have all diagonals.

Consider any unrefined model together with a refined factorisation model for it. For each $\varepsilon \subseteq \Sigma$ both T and T_ε are ε-monads, so (T_ε, T) is an ε-monad on $\mathcal{C} \times \mathcal{C}$ (and this forms a refined λ_c model on $\mathcal{C} \times \mathcal{C}$). By Theorem 4.7 we can lift (T_ε, T) to get an ε-monad $\dot{T}_\varepsilon$ on $\mathbf{LogRel}$. Moreover, each monad morphism $T_{\varepsilon \subseteq \varepsilon'}$ induces an ε-monad morphism $\dot{T}_\varepsilon \to \dot{T}_\varepsilon$. If we take the interpretations $\mathbf{LogRel}[\![b]\!]$ of base types b to be diagonal relations $\big([\![b]\!], [\![b]\!], \delta_{[\![b]\!]}\big)$, we need to interpret the constants to form a refined λ_c model on $\mathbf{LogRel}$. By the fibration's faithfulness, this interpretation is merely a property, and not a structure we need to provide. Using an inductive argument, ground types G denote diagonal relations, and if $p\,(\mathbf{LogRel}\,[\![c]\!])$ is the interpretation of the constant c in $\mathcal{C} \times \mathcal{C}$ then for all well-typed terms $\Gamma \vdash_\varepsilon M : A$ we have:

$$p\,(\mathbf{LogRel}\,[\![\Gamma \vdash_\varepsilon M : A]\!]) = ([\![\Gamma \vdash_\varepsilon M : A]\!], [\![\underline{\Gamma} \vdash M : \underline{A}]\!])$$

We use $\mathbf{LogRel}$ to compare the refined model we constructed with the original unrefined model. First:

Lemma 4.10 *Suppose that the initial ε-monad S_ε is given by the transfinite construction from §2.1.2. For each morphism $(f_1, f_2) : (1, 1) \to (T_\varepsilon\, X, T\, X)$ in $\mathcal{C} \times \mathcal{C}$, if $(f_1, f_2) : 1 \dot{\to} \dot{T}_\varepsilon\,(X, X, \delta_X)$ then $f_2 = m^{\mathbf{m}}_\varepsilon \circ f_1$.*

Under the combined assumptions of this subsection, we can now show that the refined semantics is complete for equational reasoning. For all closed terms of ground type $\vdash_\varepsilon M : G$ and $\vdash_\varepsilon N : G$,

$$[\![\, \vdash M : G]\!] = [\![\, \vdash N : G]\!] \quad \iff \quad [\![\, \vdash_\varepsilon M : G]\!] = [\![\, \vdash_\varepsilon N : G]\!]$$

To prove it, noting that ground types are interpreted as diagonal relations, we apply Lemma 4.10 to both $\mathbf{LogRel}\,[\![M]\!]$ and $\mathbf{LogRel}\,[\![N]\!]$ to show that

$$[\![\, \vdash M : G]\!] = n \circ [\![\, \vdash_\varepsilon M : G]\!] \qquad [\![\, \vdash N : G]\!] = n \circ [\![\, \vdash_\varepsilon N : G]\!]$$

Now the result follows from the fact that every $\mathcal{M}$-morphism is a monomorphism.

5 Examples

Before we conclude, we apply the factorisation construction to several examples.

Example 5.1 Continuing the global state example from §3.3, we have the full factorisation:

$$T_\emptyset = \mathrm{Id} \qquad T_{\{\mathrm{get}\}} = \mathbb{S} \Rightarrow (-) \qquad T_{\{\mathrm{set}\}} = (1 + \mathbb{Z})^{\mathbb{L}} \times (-) \qquad T_{\{\mathrm{get,set}\}} = T$$

By the completeness of the refined semantics from §4.4, we can apply the equation from §3.3 to closed programs of ground type without changing their denotations in the unrefined semantics.

Example 5.2 If instead of T we use the monad $(\mathbb{S} \Rightarrow (-) \Rightarrow R) \Rightarrow \mathbb{S} \Rightarrow R$, which combines global state with continuations (so that the language can include constants such as call/cc) then we get the same factorisation, assuming $|R| > 1$. Hence we can also verify the caching transformation in this situation. The construction in Kammar's thesis [10] does not allow this factorisation, as it is restricted to Lawevere theories, i.e., ranked monads, and T is not ranked. Note that, as call/cc is not algebraic, we cannot interpret call/cc in the refined semantics, so cannot validate transformations on subprograms that use continuations.

Example 5.3 Using the (dense, full) factorisation of $\omega\mathbf{Cpo}$, we can re-cast Kammar and Plotkin's [11] validation of effect-dependent optimisations.

Example 5.4 Let value be a base type for values of a ground type (with associated constants). Consider λ_c with a base type ref of references, and the set $\Sigma = \{\mathrm{lookup} : \mathsf{ref} \to \mathsf{value}, \mathrm{update} : \mathsf{ref} \times \mathsf{value} \to \mathbf{1}, \mathrm{alloc} : \mathsf{value} \to \mathsf{ref}\}$ of operations, so that we can read from and write to references, and allocate new references. Let $\mathbb{I}$ be the category of finite ordinals and injections between them. Plotkin and Power [26] interpret these operations the functor category $[\mathbb{I}, \mathbf{Set}]$ as follows. Let $\mathbb{V}$ be a nonempty finite set $\mathbb{V}$ of values with interpretation for the value constants. Then we interpret value as the constantly-$\mathbb{V}$ functor, and ref as the Yoneda embedding $[\![\mathsf{ref}]\!] = \mathbb{I}(1, -)$, so that $[\![\mathsf{ref}]\!]\, n$ has n elements. The local state monad is defined using a coend:

$$T\,X\,n \coloneqq \mathbb{V}^n \Rightarrow \int^{m \in \mathbb{I}} \mathbb{I}(n, m) \times \mathbb{V}^m \times Xm$$

A computation is given an initial state in $\mathbb{V}^n$, and returns an injection that describes how the original n references are distributed over the m references (so that $n \le m$), a new state in $\mathbb{V}^m$, and a result in Xm.

The category $[\mathbb{I}, \mathbf{Set}]$ has a (pointwise surjection, pointwise injection) factorisation system. For each subset $\varepsilon \in \Sigma$, since $\mathbb{V}$ is finite, we can show that the transfinite sequence S_α converges at $\aleph_0$. We can therefore show by induction on α that, for example, there are component-wise surjections from the corresponding free monads into the following functors:

$$T_{\{\mathrm{alloc}\}}\,X\,n \coloneqq \int^{m \in \mathbb{I}} \mathbb{I}(n, m) \times \mathbb{V}^{m-n} \times Xm \qquad\qquad T_{\{\mathrm{lookup},\mathrm{update}\}}\,X\,n \coloneqq \mathbb{V}^n \Rightarrow Xn \times \mathbb{V}^n$$

Calculation shows that there are pointwise injections from these into T. Theorem 2.5 (and the uniqueness of factorisations) implies they are the monads that result from factorisation. Now note that there are two sequencing morphisms $T_{\{\mathrm{alloc}\}}\,X \times T_{\{\mathrm{alloc}\}}\,Y \to T_{\{\mathrm{alloc}\}}\,(X \times Y)$, one that does the left computation first and one that does right first. It is easy to check that these are equal, i.e., $T_{\{\mathrm{alloc}\}}$ is commutative, and hence we can validate a transformation that reorders computations that only allocate.

6 Conclusion

We have presented a factorisation theorem for cutting down a monad into sub-monads based on a factorisation system. We showed how this construction gives uniform semantics for Gifford-style type-and-effect systems. Synthesising Hughes and Jacobs's characterisation of fibrations arising from factorisation systems and Katsumata's axiomatisation of fibrations for logical relations, we provide a general proof that the factorisation construction is sound and complete for effect-dependent equational reasoning.

We would like to generalise the completeness theorem to programs of higher-order types, and not just ground types. Reynolds [29] relates direct and continuation semantics by defining domain-theoretic partial maps between the two semantics, and proves such a theorem. Felleisen and Cartwright [5] provide an analogous construction and proof for free effects and their handlers [27,1], but their semantics does not involve monads. Well-powered factorisation systems for logical relations induce categories of partial maps via Fiore's axiomatic domain theory [6]. The axiomatic development is particularly appealing because factorisation systems of interest, such as the (dense, full) factorisation of $\omega\mathbf{Cpo}$ do not admit a representation using a lifting monad.

We want to relate the free lifting to other lifting techniques, most notably $\top\top$-, and codensity-, lifting. We would also like to relate Benton et al.'s [2] relational models to our construction. We want to apply this construction to more sophisticated computational effects, such as dynamic memory allocation [9]. Another application area to the free lifting is relational parametricity with effects — we have used it as a semantic precursor to the more syntactic work on analysing the value restriction [12], and we hope it applies more widely. Finally, there is still a wide gap between Gifford-style type-and-effect systems and the full generality of graded monads. We hope our account would carry over to such settings.

Acknowledgements. This work has been supported by an Engineering and Physical Sciences Research Council (ESPRC) studentship, EPSRC grants EP/N007387/1 'Quantum computation as a programming language', EPSRC Leadership Fellowship EP/H005633/1 'Semantic Foundations for Real-World Systems', Institute for Information & Communications Technology Promotion (IITP) grant funded by the Korea government (MSIP) No. R0190-16-2011 'Development of Vulnerability Discovery Technologies for IoT Software Security', a Balliol College Oxford Career Development Fellowship, European Research Council Grant 'Events, Causality and Symmetry — the next generation semantics', and Isaac Newton Trust grant 'algebraic theories, computational effects, and concurrency'. We would like to thank Marcelo Fiore, Mathieu Huot, Justus Matthiesen, and Ian Orton for fruitful discussions and suggestions.

References

[1] Andrej Bauer and Matija Pretnar. Programming with algebraic effects and handlers. *Journal of Logical and Algebraic Methods in Programming*, 84(1):108 – 123, 2015. Special Issue: The 23rd Nordic Workshop on Programming Theory (NWPT 2011) Special Issue: Domains X, International workshop on Domain Theory and applications, Swansea, 5-7 September, 2011.

[2] Nick Benton, Martin Hofmann, and Vivek Nigam. Effect-dependent transformations for concurrent programs. *Science of Computer Programming*, 155:27 – 51, 2018. Selected and Extended papers from the International Symposium on Principles and Practice of Declarative Programming 2016.

[3] Nick Benton, Andrew Kennedy, and George Russell. Compiling Standard ML to Java Bytecodes. In *Proceedings of the Third ACM SIGPLAN International Conference on Functional Programming*, ICFP '98, pages 129–140, New York, NY, USA, 1998. ACM.

[4] Aldridge K. Bousfield. Constructions of factorization systems in categories. *Journal of Pure and Applied Algebra*, 9(2):207–220, 1977.

[5] Robert Cartwright and Matthias Felleisen. *Extensible denotational language specifications*, pages 244–272. Springer Berlin Heidelberg, Berlin, Heidelberg, 1994.

[6] Marcelo Pablo Fiore. *Axiomatic Domain Theory in Categories of Partial Maps*. Distinguished Dissertations in Computer Science. Cambridge University Press, 1996.

[7] Jesse Hughes and Bart Jacobs. Factorization systems and fibrations. *Electronic Notes in Theoretical Computer Science*, 69:156 – 182, 2003.

[8] Bart Jacobs. *Categorical Logic and Type Theory*. Number 141 in Studies in Logic and the Foundations of Mathematics. North Holland, Amsterdam, 1999.

[9] O. Kammar, P. B. Levy, S. K. Moss, and S. Staton. A monad for full ground reference cells. In *2017 32nd Annual ACM/IEEE Symposium on Logic in Computer Science (LICS)*, pages 1–12, June 2017.

[10] Ohad Kammar. *Algebraic theory of type-and-effect systems*. PhD thesis, University of Edinburgh, UK, 2014.

[11] Ohad Kammar and Gordon D. Plotkin. Algebraic foundations for effect-dependent optimisations. In *Proceedings of the 39th Annual ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages*, POPL '12, pages 349–360, New York, NY, USA, 2012. ACM.

[12] Ohad Kammar and Matija Pretnar. No value restriction is needed for algebraic effects and handlers. *Journal of Functional Programming*, 27:e7, 2017.

[13] Shin-ya Katsumata. A semantic formulation of $\top\top$-lifting and logical predicates for computational metalanguage. In Luke Ong, editor, *Computer Science Logic*, pages 87–102, Berlin, Heidelberg, 2005. Springer Berlin Heidelberg.

[14] Shin-ya Katsumata. Relating computational effects by $\top\top$-lifting. *Inf. Comput.*, 222:228–246, 2013.

[15] Shin-ya Katsumata. Parametric effect monads and semantics of effect systems. *SIGPLAN Not.*, 49(1):633–645, 2014.

[16] Shin-ya Katsumata and Tetsuya Sato. Codensity liftings of monads. In *CALCO*, 2015.

[17] G.M. Kelly. A unified treatment of transfinite constructions for free algebras, free monoids, colimits, associated sheaves, and so on. *Bulletin of the Australian Mathematical Society*, 22(1):1–83, 1980.

[18] G.M. Kelly. Two addenda to the author's transfinite constructions. *Bulletin of the Australian Mathematical Society*, 26(2):221237, 1982.

[19] Daniel Lehmann and Ana Pasztor. Epis need not be dense. *Theoretical Computer Science*, 17(2):151 – 161, 1982.

[20] John M. Lucassen and David K. Gifford. Polymorphic effect systems. In *Proceedings of the 15th ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages*, POPL '88, pages 47–57, New York, NY, USA, 1988. ACM.

[21] Saunders Mac Lane. *Categories for the Working Mathematician (Graduate Texts in Mathematics)*. Springer, 2nd edition, 1998.

[22] Francisco Marmolejo and Richard J. Wood. Monads as extension systems — no iteration is necessary. *Theory and Applications of Categories*, 24(4):84–113, 2010.

[23] José Meseguer. Completions, factorizations and colimits for ω-posets. In *Mathematical Logic in Computer Science, Salgotarjan, 1978, Colloquia Mathematica Societatis Janos Bolyai*, volume 26, pages 509–545. North Holland, 1981.

[24] Eugenio Moggi. Computational lambda-calculus and monads. In *Proceedings of the Fourth Annual Symposium on Logic in Computer Science*, pages 14–23, Piscataway, NJ, USA, 1989. IEEE Press.

[25] Gordon Plotkin and John Power. Algebraic operations and generic effects. *Applied Categorical Structures*, 11(1):69–94, 2003.

[26] Gordon D. Plotkin and John Power. Notions of computation determine monads. In *Proceedings of the 5th International Conference on Foundations of Software Science and Computation Structures*, pages 342–356, London, UK, 2002. Springer-Verlag.

[27] Gordon D. Plotkin and Matija Pretnar. Handlers of algebraic effects. In *Proceedings of the 18th European Symposium on Programming Languages and Systems: Held As Part of the Joint European Conferences on Theory and Practice of Software, ETAPS 2009*, ESOP '09, pages 80–94. Springer-Verlag, Berlin, Heidelberg, 2009.

[28] A John Power. Enriched Lawvere theories. *Theory and Applications of Categories*, 6(7):83–93, 1999.

[29] John C. Reynolds. On the relation between direct and continuation semantics. In Jacques Loeckx, editor, *Automata, Languages and Programming*, pages 141–156, Berlin, Heidelberg, 1974. Springer.

[30] Andrew P. Tolmach. Optimizing ML using a hierarchy of monadic types. In *Types in Compilation*, pages 97–115, 1998.

[31] Philip Wadler. The marriage of effects and monads. *SIGPLAN Not.*, 34(1):63–74, 1998.