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

Ohad Kammar
Department of Computer Science
University of Oxford
Oxford, England

Dylan McDermott
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) = \_{-} : unit . 3*(get \ell)
2 in \ell := 1;
3 \ell := triple() + triple()

The locally-defined function triple : unit -> int triples the value read from memory location \ell. 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 \ell := 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 \ell’ 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

This paper is electronically published in
Electronic Notes in Theoretical Computer Science
URL: www.elsevier.nl/locate/entcs
factorised as

\[ A \xrightarrow{\alpha, \beta} B \]

2.1.1 Factorisation systems

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 [22].

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 [22].

2.1.1 Factorisation 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

\[ \lambda_{\cdot} : \text{unit} \to \text{int}. \ell' := (1 + \text{get} \ \ell'); \ 3*(\text{get} \ \ell) \]

Type-and-effect systems [21] refine types, such as \( \text{triple} : \text{unit} \rightarrow \text{int} \), to propagate the information about which computational effects code pieces may invoke, e.g., decorating function types with additional effect annotations:

\[ \text{triple} : \text{unit} \rightarrow \to \text{int} \]

In Gifford-style systems, these annotations are finite sets of effect operations, such as \( \varepsilon := \{ \text{get}, \text{set} \} \). For example, for every proper subset \( \varepsilon \subset \{ \text{get}, \text{set} \} \), the caching transformation for every function \( f : A \xrightarrow{\varepsilon} B \) is semantics preserving, while for \( \varepsilon = \{ \text{get}, \text{set} \} \) 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 Toltach [31], Wadler [32], and Benton et al. [3], and continuing to this day [2]. In their most general form, the semantics of an effect system consists of a graded monad [15], a compatible family of monad-like structures \( T \_ \varepsilon \) indexed by the effect annotations \( \varepsilon \).

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 \( T \_ \varepsilon \), 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 arising from 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 [26]. While Kammar [10] describes it in the special set-theoretic case, we believe this folklore monadic lifting methodology 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 [29] 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 \( \lambda \)-calculus. Section 4 instruments a logical relations soundness and completeness proof from the factorisation construction. Section 5 surveys example applications of 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 [22].
\( m : f[A] \to B \) of this image into \( f^\prime \)'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 \( e : A \to B \) for an \( \mathcal{E} \)-morphism and \( m : B \to C \) 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 \( h \circ e = f \) and \( m \circ h = g \), as on the right:

\[
\begin{array}{ccc}
W & \xrightarrow{e} & X \\
\downarrow_{f} & & \downarrow_{g} \\
Y & \xrightarrow{m} & Z
\end{array}
\implies
\begin{array}{ccc}
W & \xrightarrow{e} & X \\
\downarrow_{f} & = & \downarrow_{g} \\
Y & \xrightarrow{m} & Z
\end{array}
\]

Under the first two axioms, the diagonal fill-in axiom is equivalent to a form of functoriality in factorisation, as in the following diagram:

\[
\begin{array}{ccc}
X & \xrightarrow{f} & Y \\
\downarrow_{g} & = & \downarrow_{g_\omega} \\
X' & \xrightarrow{f'} & Y'
\end{array}
\implies
\begin{array}{ccc}
X & \xrightarrow{e} & A \\
\downarrow_{g_\omega} & = & \downarrow_{g_\omega} \\
X' & \xrightarrow{e'} & A'
\end{array}
\]

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 \( \text{Set} \) 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 [24]] Consider the category \( \omega \text{Cpo} \) of partial orders possessing all least upper bounds (lubs) of \( \omega \)-indexed monotone sequences (\( \omega \)-chains), i.e., \( \omega \)-cpos, and monotone functions between them preserving these lubs, i.e., Scott-continuous functions. A *dense function* is a continuous function \( e : X \to Y \) such that the smallest \( \omega \)-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 \( f : X \to Y \) such that \( mx \leq mx' \) implies \( x \leq x' \) for each \( x \in X \). The category \( \omega \text{Cpo} \) has (dense, full) as a factorisation system. The full functions form a proper subclass of the monos, and the dense functions form a proper subclass of the epis [20].

**Example 2.4** Consider the functor category \([W, \mathcal{C}]\), for a small category \( W \) 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 \([W, \mathcal{C}]\).

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 \( f = (A_1^f, A_2^f, f) \) consisting of a morphism \( f : A_1^f \to A_2^f \), and whose morphisms \( h : f \to g \) are pairs \((h_1, h_2)\) consisting of morphisms \( h_1 : A_1^f \to A_1^g \) 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 do not need any semantic properties such as 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 \((T, \text{return}^T, \mu^T)\) where:

- the *functor part* \( T \) assigns to every \( \mathcal{C} \)-object \( A \) another \( \mathcal{C} \)-object \( TA \), and to every \( \mathcal{C} \)-morphism \( f : A \to B \) another \( \mathcal{C} \)-morphism \( Tf : TA \to TB \);
- the *unit* \( \text{return}^T \) assigns to every \( \mathcal{C} \)-object \( A \) a \( \mathcal{C} \)-morphism \( \text{return}^T_A : A \to TA \); and
- the *multiplication* \( \mu^T \) assigns to every \( \mathcal{C} \)-object \( A \) a \( \mathcal{C} \)-morphism \( \mu^T_A : T^2A \to TA \).

3
A monad is thus a monad structure $T$ satisfying the well-known monad laws [22, Section 7.1]. When $C$ has finite products, a strong monad structure is a monad structure $T$ with an additional structure component:

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

A strong monad [19] is thus a strong monad structure satisfying the well-known laws. We similarly define Kleisli triple structures $T = (\mathcal{T}, \text{return}^T, \Rightarrow^T)$, demanding only an assignment $\mathcal{T}$ on objects, morphisms $\text{return}^T_A : A \to TA$, and a Kleisli extension $\Rightarrow^T_{A,B} : TA \to TB$ for every $f : A \to TB$. Finally, when $C$ is cartesian closed, we define a strong Kleisli triple structure $T = (\mathcal{T}, \text{return}^T, \Rightarrow^T)$ analogously, replacing $\Rightarrow$ with an assignment of a morphism $\Rightarrow^T_{A,B} : TA \times TB \to TB$ for every pair of $C$-objects $A$ and $B$. This more general Kleisli extension induces morphisms $\text{str}^T_{A,B} : A \times TB \to T(A \times B)$. (Strong) Kleisli triples are (strong) Kleisli triple structures satisfying additional laws.

Morphisms $m : S \to T$ of the above structures are natural transformations $m : S \to T$ that preserve the structure, i.e., satisfy the same conditions a (strong) monad morphism should. Such morphisms provide categories of (strong) monad structures and of (strong) Kleisli triple structures. They also provide the subcategories of (strong) monads and of (strong) Kleisli triples. There are isomorphisms between the categories of (strong) monads and (strong) Kleisli triples. The monad laws are necessary to establish this: the isomorphisms fail to extend to isomorphisms between the structure categories.

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

- the carrier $A$, a $C$-object; and
- the algebra map $\text{alg}_A$, a $C$-morphism $\text{alg}_A : TA \to A$.

When $T$ is a monad, an algebra is an algebra structure satisfying the well-known algebra properties [22, Section 7.2]. Similarly, when $T$ is a Kleisli triple structure, an algebra structure $A = (A, \Rightarrow_A)$ replaces the algebra map with:

- the extension operator $\Rightarrow_A$ which assigns to every morphism $f : X \to A$ a morphism $\Rightarrow f : TX \to A$.

When $T$ is a Kleisli triple, an algebra is an algebra structure satisfying [23], for every $f : X \to A$, and every $g : X \to TY$, $h : Y \to A$:

$$
\begin{array}{ccc}
X & \xrightarrow{f} & TX \\
\downarrow \scriptstyle \text{return} & & \downarrow \scriptstyle \Rightarrow f \\
A & \Rightarrow_A & A
\end{array}
$$

Similarly, we define an algebra structure for a strong Kleisli triple structure by replacing the extension operator with an internal extension operator $\Rightarrow : TX \times AX \to A$, and algebras for a strong Kleisli triple internalise the two equations above.

Let $T$ be a monad over a category $C$. Recall that a Kleisli arrow is a morphism $f : A \to TB$. When $C$ is cartesian closed and $T$ is strong, an algebraic operation [26] $\alpha : A \to B$ for $T$ assigns to every $C$-object $X$ a $C$-morphism $\alpha_X : (TX)^B \to (TX)^A$, natural in $X$, and respecting the multiplication/extension and the strength. Plotkin and Power [26] establish a bijection between Kleisli arrows $f : A \to TB$ and algebraic operations $\alpha : A \to B$ given by:

$$
\begin{array}{ccc}
\text{uncurry} \alpha_X : A \times (TX)^B & \xrightarrow{f \times \text{id}} & TB \times (TX)^B \\
\downarrow \scriptstyle \Rightarrow \end{array}
$$

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

$$
\begin{array}{ccc}
X \times F(TY) & \xrightarrow{\text{str}^F} & F(X \times TY) \\
\downarrow \scriptstyle \text{id} \times \beta & = & \downarrow \scriptstyle \beta \\
X \times TY & \xrightarrow{\text{str}^F} & T(X \times Y)
\end{array}
$$

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

An effect signature $\varepsilon$ in a category $C$ consists of a set $\varepsilon$ of operations and an $\varepsilon$-indexed family of pairs of
$\mathcal{C}$-objects. We write $(op : X \to Y) \in \varepsilon$ when $op \in \varepsilon$ and $(X, Y)$ is the $op$-th component in $\varepsilon$. Here $X$ is thought of as the input to the operation, and $Y$ as the output. We write $\varepsilon \subseteq \varepsilon'$ when $\varepsilon$ is a subset of $\varepsilon'$ and both agree component-wise.

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

### 2.2 The factorisation theorems

Let $(\mathcal{E}, \mathcal{M})$ be a factorisation system for a category $\mathcal{C}$, and let $S$ be a monad structure on $\mathcal{C}$. We say that $(\mathcal{E}, \mathcal{M})$ is closed under $S$ when, for every $e : A \to B$ in $\mathcal{E}$, we have $Se : SA \to SB$ 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 $m : m[S] \to T$ by choosing a factorisation for each $m_X$, setting for each $f : X \to Y$, and $Z$:

\[
\begin{array}{c}
SX \xrightarrow{m_X} TX \\
\downarrow m^e_X \quad \downarrow m^m_X\ \\
S[X]X \xrightarrow{m^e_X} S[X]Y \xrightarrow{m^m_Y} T(X)Y \xrightarrow{m^m_{X,Y}} T(X(Y)) \\
\end{array}
\]

This definition makes $m^e : S \to m[S]$ a monad structure morphism with components in $\mathcal{E}$, and $m^m : m[S] \to T$ 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 of $A$ by setting:

\[
\alg^T_A : SA \xrightarrow{m^e_A} m[S]A \xrightarrow{m^m_{A}} A
\]

When $\mathcal{C}$ has finite products, we say that the 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 $m : S \to T$ through $m[S]$ as on the left:

\[
\begin{array}{c}
X \times SY \xrightarrow{id \times m^m_{Y}} X \times m[S]Y \\
\downarrow \str_{X,Y} \quad \downarrow \str_{X,Y}^m \quad \downarrow \str_{X,Y}^m \quad \downarrow \str_{X,Y}^m \quad \downarrow \str_{X,Y}^m \\
S(X \times Y) \xrightarrow{m^e_{X,Y}} X \times TY \xrightarrow{m^m_{X,Y}} m[S](X \times Y) \xrightarrow{m^m_{X,Y}} T(X \times Y) \\
\end{array}
\]

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 $m^e$ and $m^m$ are monad morphisms. As a consequence, every algebra for $m[S]$ induces an algebra for $S$.  

5
Using the diagonal fill-in property, we can 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 (Factoriality)** Let \((\mathcal{E}, M)\) be a factorisation system for a category \( C \), and let \( f : (S, T, m) \to (S', T', m') \) be a commuting square of monad structure morphisms. If \((\mathcal{E}, M)\) 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 \((\mathcal{E}, M)\) 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 \((\mathcal{E}, M)\). When it is an *epi-mono* factorisation system, i.e., a pair \((\mathcal{E}, M)\) in which \( M \) 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 \( F_x \).

We recall Kelly’s [17,18] transfinite construction of the free \( F \)-monad when \( C \) has \( \kappa \)-directed colimits and \( F \) is an arbitrary functor that is \( \kappa \)-ranked, i.e., preserves these colimits, for some regular cardinal \( \kappa \). Define an ordinal-indexed sequence of functors \( S_\alpha : C \to C \) by transfinite induction on \( \alpha \) as follows:

\[
S_0 := \text{Id} \quad \quad S_{\alpha+1} := \text{Id} + F \circ S_\alpha \quad \quad S_\lambda := \operatorname{colim}_{\alpha < \lambda} S_\alpha \quad (\lambda \text{ a limit ordinal})
\]

Each colimit is directed: the diagram includes morphisms \( S_\alpha \to S_\alpha' \) for \( \alpha < \alpha' \); these morphisms are defined by transfinite recursion. The free monad \( S_F \) for \( F \) then has underlying endofunctor \( S_F := \operatorname{colim}_{\alpha < \kappa} S_\alpha \). If \( F \) is also strong then \( S_F \) is the initial object of \( F \)-\text{Mnd}\( C \).

To apply the factorisation theorem, we need the free monad to be compatible with the factorisation system. We give a sufficient condition on \( F \) for compatibility.

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

To apply the last lemma to the signature functor \( F_x \), we want to show that \( F_x \) preserves \( \kappa \)-directed colimits for some \( \kappa \), and that \( E \) is closed under \( F_x \). For colimit preservation, the following lemma covers our examples.

**Lemma 2.8** Let \( \varepsilon \) be an effect signature in a locally presentable cartesian closed category \( C \). Then the functor \( F_\varepsilon \) preserves \( \kappa \)-directed colimits for some regular cardinal \( \kappa \).

However, some \( F_x \) may be incompatible with some factorisation systems, since exponentials might not preserve \( \varepsilon \)-morphisms:
(e : A) ∈ K
Γ ⊢ e : A
Γ ⊢ op : A → B ∈ Σ
(Γ, e, M : A ⊢ (op : A → B)) ∈ Σ
Γ ⊢ e, M : A ∈ Σ
Γ ⊢ ε, M : A ∈ Σ
(ε ⊆ ε')
(Γ, e, x : A, M : A) ∈ Σ

Example 2.9 Consider the (dense, full) factorisation system on ωCpo. Exponentials (-)^Y preserve dense maps iff Y is a countable discrete ω-cpo. For a simple illustration, take the discrete natural numbers N and the ordinal ω + 1. Take Y := ω + 1, and consider the inclusion e : N → ω + 1, which is a dense map. Every monotone function f : ω + 1 → N is constant, and so the ω-chain-closure of e^Y[N] contains only constant functions. Therefore, the identity function x := id ∈ (ω + 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 [25] computational λ-calculus, λ_e, 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_e is parametrised by three sets: a set B of base types ranged over by b; a set Σ of operations ranged over by op; and a set K of constants ranged over by e. The metavariable x ranges over some set of variables and ε ranges over finite subsets of Σ. The syntax of types A, B (base types, products and sums, and function types), ground types G, and terms M of the λ_e-calculus is given as follows:

\[
\begin{align*}
A, B & ::= b \mid 1 \mid A \times B \mid 0 \mid A + B \mid A \xrightarrow{e} B \\
G & ::= b \mid 1 \mid G_1 \times G_2 \mid 0 \mid G_1 + G_2 \\
M, N & ::= c \mid \text{op}_M \mid x \mid () \mid (M, N) \mid \text{fst}_M \mid \text{snd}_M \mid \text{elim}_M \mid \text{inl}_M \mid \text{inr}_M \mid \text{match}_M \mid \text{λx}_M \mid M \ N
\end{align*}
\]

The main difference to Moggi’s calculus is that we include a specified set of constructs 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_e’s type system, we need some typing information for effect operations and the constants. Formally, a λ_e signature is a triple (B, Σ, K) consisting of: a set B of base types; a family of pairs of ground types Σ indexed by a set of operations Σ; and a family of types K indexed by a set of constants K. We write c : A when the type A is the c-component of K, and op : G → G’ when (G, G’) is the op-component of Σ.

Given a λ_e signature we define two type systems. The type-and-effect system consists of a typing judgment Γ ⊢ e, M : A given inductively by the rules in Figure 1. Such judgments 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 ε ⊆ Σ. The rules are standard for such systems.

We recover the usual type system for λ_e 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 A, and similarly for contexts Γ. The unrefined typing judgments Γ ⊢ M : 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 Γ ⊢ e, M : A then Γ ⊢ M : A.
3.2 Semantics

Fix a \( \lambda_c \) signature \((B, \Sigma, \mathcal{K})\). Recall that a bicartesian closed category is a cartesian closed category with finite coproducts. Given a bicartesian closed category \( C \) and an object \([b] \in C\) for each \( b \in B\), we can define the interpretation \([G]\) of each ground type \( G \) in the usual way, and interpret each \( c \in \Sigma \) with operations \( op: G \to G' \) as an effect signature \([\varepsilon]\) with operations \( op: [G] \to [G']\).

An \textit{unrefined} \( \lambda_c \) \textit{model} consists of: a bicartesian closed category \( C \); an object \([b] \in C\) for each \( b \in B\); a \([\Sigma]\)-monad \( T \) on \( C \) (recall the definition of \( \varepsilon \)-monad from Section 2.1.2); and a morphism \([c]: 1 \to [A]\) for each constant \((c: A) \in \mathcal{K}\). Unrefined models interpret the unrefined judgments \( \Gamma \vDash M : A \), with types and contexts denoting \( C \)-objects \([B]\) and \([\Gamma]\), and judgments denoting Kleisli arrows \([\Gamma \vDash M : A : [\Gamma] \to T[A]\).

To interpret type-and-effect judgments in their greatest generality, one replaces the monad with a \textit{refined} \( \lambda_c \) \textit{interpretation} of \( \varepsilon \)-monad \( [\varepsilon]: \Lambda \to \text{Set} \). Third, we assume a \textit{refined} \( \lambda_c \) \textit{interpretation} of base types. For our \( \lambda_c \) \textit{signature} \((\mathcal{B}, \Sigma, \mathcal{K})\), we set \( \mathcal{K} := \{\text{int} \times \text{int} \to \text{int}\} \cup \{\ell : \text{loc} | \ell \in \mathcal{L}\} \cup \{a : \text{int} | a \in \mathbb{Z}\}

The only difference between the two model structures is the functorial assignment \( \varepsilon \)-\textit{monad} \( \varepsilon \) \textit{morphism}, which replaces the monad with a \textit{refined} \( \lambda_c \) \textit{model} structure. 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 \( \lambda_c \) \textit{signature}, we take: \( \mathcal{B} := \{\text{loc}, \text{int}\}, \Sigma := \{\text{get} : \text{loc} \to \text{int}, \text{set} : \text{loc} \times \text{int} \to 1\}, \) and 

\[
\mathcal{K} := \{+ : \text{int} \times \text{int} \to \text{int}\} \cup \{\ell : \text{loc} | \ell \in \mathcal{L}\} \cup \{a : \text{int} | a \in \mathbb{Z}\}
\]

For the unrefined model structure, we interpret: \([\text{loc}]:= \mathbb{L}\) and \([\text{int}]:= \mathbb{Z}\). For our monad, we set \( S := \mathbb{Z}^{\mathbb{L}} \) and take \( T \) to be the \( S \)-state monad, \( TX := (S \times X)^S \), with the usual interpretation for \text{get} and \text{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 \( \text{Set} \). We can calculate that \( T_{\{\text{set}\}} X = (\mathbb{L} + \mathbb{Z})^X \times X \) is the writer monad for the following \textit{overwriting monoid} \((\mathbb{L} + \mathbb{Z})^X, 1, \ast)\): 

\[
1 := (\ell \ast x)_{\ell \in \mathbb{L}}, \quad ((a \ell)_{\ell \in \mathbb{L}} \ast (b \ell)_{\ell \in \mathbb{L}}) \ell = \begin{cases} b \ell & \text{if } b \ell \neq \ell \ast \ast \\ a \ell & \text{otherwise} \end{cases}
\]

I.e., an injected unit value at location \( \ell \) 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 \{\text{set}\}-monad is the smallest set satisfying \( S_{\{\text{set}\}} X \cong X + \mathbb{L} \times \mathbb{Z} \times S_{\{\text{set}\}} X \). The unique \{\text{set}\}-monad morphism \( m_{\{\text{set}\}} : S_{\{\text{set}\}} \to T \) satisfies:

\[
m_{\{\text{set}\}}(\ell x) := \lambda s. (s x) \quad m_{\{\text{set}\}}(\ell a, r) := \lambda s. (s \ell \mapsto a), m_{\{\text{set}\}}(r)
\]

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

\[
m_{\{\text{set}\}}^e(\ell x) \mapsto (\ell \ast x) \quad m_{\{\text{set}\}}^e(\ell a, r) \mapsto ((\ell \ast x)_{\ell \in \mathbb{L}} \ast (-)) \times \text{id}(m_{\{\text{set}\}}(r))
\]

We then interpret \(+\) as addition, as \( T_0 \) 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 \( \Gamma \vDash_{\{\text{set}\}} M : \text{int} \).
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. We then describe the free lifting of monads to logical relations, and use this to prove the completeness of the refined semantics (Theorem 4.12).

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

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

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

For each \( f : I \to J \) in \( C \) there is an inverse image functor \( f^* : D_J \to D_I \) that sends an object \( X \) to an object \( Y \) such that \( f : X \Rightarrow Y \) is Cartesian. The object \( Y \) is unique up to isomorphism in \( D_I \); for any \( Y' \) with the same property we have \( Y \equiv Y' \). We will also postulate that \( f^* \) has a left adjoint \( f_* : D_I \to D_J \), 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 bicartesian closed structure. For example, products in \( D \) 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.

Definition 4.1 A fibration for logical relations [14] over a bicartesian closed category \( C \) is a faithful fibration \( p : D \to C \) such that:

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

Our only deviation from Katsumata’s definition is to not require fibres to be partial orders, due to our use of non-strict factorisation systems. Since the fibration is faithful, fibres are preorders.

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

Lemma 4.2 (Katsumata [14, Proposition 6]) Let \( p : D \to C \) be a fibration for logical relations, and let \( F : C' \to C \) be a product-preserving functor. The projection from the pullback \( F^* p \) of \( p \) along \( F \) is a fibration for logical relations on \( C' \).

\[
\begin{array}{ccc}
F^* D & \longrightarrow & D \\
\downarrow F^* p & & \downarrow p \\
C' & \longrightarrow & C
\end{array}
\]

When we choose the product functor \( F := (\times) : C \times C \to C \), we call \( F^* D \) the category of binary logical \( p \)-relations over \( C \).
4.2 Fibrations from factorisation systems

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

\[
\begin{array}{c}
X \ar[r]^f & X' \\
\downarrow m & \downarrow \quad \downarrow m' \\
Y \ar[r]_f & Y'
\end{array}
\]

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

**Example 4.3** Consider the (surjection, injection) factorisation for \(\text{Set}\). Every injection \(m : X \to Y\) is equal to the composition of an inclusion \(i\) and an isomorphism. In this case, we have \(m = i\). 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 4.4** Similarly for the (dense, full) factorisation for \(\omega\text{Cpo}\), the full functions are the chain-closed subsets. Inverse images are the usual inverse images, but direct images are now the \(\omega\)-chain-closure of the direct image.

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

**Definition 4.5** [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:

- 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 \(\text{cod}\) 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 \(n' : X' \to Y'\) the canonical morphism \(X \Rightarrow X' \Rightarrow X \Rightarrow Y'\) is an \(\mathcal{M}\)-morphism, and hence that \(\mathcal{M}\) has exponentials \(m \Rightarrow m'\), which are given by the following pullback:

\[
\begin{array}{c}
Z \ar[r]^j & X \ar[r]^f & X' \\
\downarrow m & \downarrow & \downarrow m' \\
Y \ar[r]_{n'} & Y' \ar[r]_{m} & X \ar[r]_{f} & Y'
\end{array}
\]

**Lemma 4.6** Let \((\mathcal{E}, \mathcal{M})\) be a factorisation system over a bicartesian closed category \(\mathcal{C}\). The codomain functor \(\text{cod} : \mathcal{M} \to \mathcal{C}\) 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 factorisation system induced by Hughes and Jacobs’s correspondence is a factorisation system for logical relations.

**Example 4.7** The factorisation systems (surjection, injection) for \(\text{Set}\) and (dense, full) for \(\omega\text{Cpo}\) 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 \( p : D \to C \) be a faithful fibration, \( \varepsilon \) be an effect signature in \( D \), and \( T \) be a \( p \varepsilon \)-monad on \( C \), where \( p \varepsilon \) is the effect signature with operations \( op : pX \to pY \) for \( (op : X \to Y) \in \varepsilon \). A lifting of \( T \) to \( D \) is an \( \varepsilon \)-monad \( \hat{T} \) on \( D \) such that:

- for each \( X \in D \) we have \( p(\hat{T}X) = T(pX) \);
- the multiplication lifts: \( p(\mu \hat{T}) = \mu T \);
- for each \( f : X \to Y \) we have \( p(\hat{T}f) = T(f) \);
- the strength lifts: \( p(str \hat{T}) = str T \); and
- the unit lifts: \( p(\text{return} \hat{T}) = \text{return} T \).

Only the object action of \( \hat{T} \) 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 \( T \uplus T \)-lifting [13] and the codensity lifting [16]. We instead use the free lifting, which is the \( \varepsilon \)-monad that is initial amongst all \( \varepsilon \)-liftings. The proof of completeness relies on initiality. The construction of the free lifting is folklore, and is described for binary relations over \( \text{Set} \) in Kammar’s thesis [10]. We describe it for the general case of a fibration for logical relations here.

Let \( p : D \to C \) 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 \( X \in D \) define \( RX \) as the set of all \( X' \) in the representing set of \( D_{\Sigma}(pX) \) such that:

- The unit respects \( X' \): \( return^T : X \to X' \).
- For each \( (op : A \to B) \in \varepsilon \) the algebraic operation \( \alpha_{op} \) respects \( X' \): \( \alpha_{op} : B \Rightarrow X' \Rightarrow A \Rightarrow X' \), where \( \Rightarrow \) denotes exponentials in \( D \).

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 \( RX \) can be thought of as candidates for \( \hat{T}X \). We define the free lifting of \( T \) to \( D \) on objects by \( \hat{T}X \coloneqq \bigwedge RX \), so that \( \hat{T}X \) is the least element of \( RX \) with respect to the order \( \leq \) on the fibre. This definition extends uniquely to a lifting of \( T \) to \( D \).

**Theorem 4.8** \( \hat{T} \) is a lifting of \( T \) to \( D \), and is initial: for all liftings \( T' \), the identity lifts to a (necessarily unique) \( \varepsilon \)-monad morphism \( \hat{T} \to T' \).

4.4 Completeness

We now return to the language \( \lambda_\varepsilon \) and relate the refined semantics we construct at the end of Section 3.2 with the unrefined semantics. Suppose that the factorisation system we used to construct the refined semantics is a \( \lambda \varepsilon \)-lifting \([13]\) and the codensity lifting \([16]\). We instead use the free lifting, which is the \( \varepsilon \)-monad that is initial amongst all \( \varepsilon \)-liftings. The proof of completeness relies on initiality. The construction of the free lifting is folklore, and is described for binary relations over \( \text{Set} \) in Kammar’s thesis [10]. We describe it for the general case of a fibration for logical relations here.

Let \( p : D \to C \) 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 \( X \in D \) define \( RX \) as the set of all \( X' \) in the representing set of \( D_{\Sigma}(pX) \) such that:

- The unit respects \( X' \): \( return^T : X \to X' \).
- For each \( (op : A \to B) \in \varepsilon \) the algebraic operation \( \alpha_{op} \) respects \( X' \): \( \alpha_{op} : B \Rightarrow X' \Rightarrow A \Rightarrow X' \), where \( \Rightarrow \) denotes exponentials in \( D \).

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 \( RX \) can be thought of as candidates for \( \hat{T}X \). We define the free lifting of \( T \) to \( D \) on objects by \( \hat{T}X \coloneqq \bigwedge RX \), so that \( \hat{T}X \) is the least element of \( RX \) with respect to the order \( \leq \) on the fibre. This definition extends uniquely to a lifting of \( T \) to \( D \).

**Theorem 4.8** \( \hat{T} \) is a lifting of \( T \) to \( D \), and is initial: for all liftings \( T' \), the identity lifts to a (necessarily unique) \( \varepsilon \)-monad morphism \( \hat{T} \to T' \).

**Example 4.9** The factorisation systems (surjection, injection) over \( \text{Set} \) and (dense, full) over \( \omega \text{Cpo} \) are well-powered and have all diagonals. For every factorisation system \( (\mathcal{E}, \mathcal{M}) \) for \( C \) and every small category \( W \), 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 4.10** Over \( \text{Set} \), the factorisation system (iso, any) is not well-powered, and the factorisation system (any, iso) does not have all diagonals.

Consider any logical relations model together with a refined factorisation model for it. For each \( \varepsilon \subseteq \Sigma \) both \( T \) and \( \varepsilon \)-monads, so \( (T_\varepsilon, T) \) is an \( \varepsilon \)-monad on \( \mathcal{C} \times \mathcal{C} \) (and this forms a refined \( \lambda_\varepsilon \) model on \( \mathcal{C} \times \mathcal{C} \)). By Theorem 4.8 we can lift \( (T_\varepsilon, T) \) to get an \( \varepsilon \)-monad \( \hat{T}_\varepsilon \) on \( \text{LogRel} \). Moreover, each monad morphism \( T_{\varepsilon, \varepsilon'} \) induces an \( \varepsilon \)-monad morphism \( \hat{T}_{\varepsilon, \varepsilon'} \to \hat{T}_{\varepsilon'} \), since \( \hat{T}_{\varepsilon, \varepsilon'} \) is initial and \( (T_{\varepsilon, \varepsilon'})(\hat{T}_\varepsilon X) \Rightarrow \hat{T}_\varepsilon X \). If we take the interpretations \( \text{LogRel}[b] \) of base types \( b \) to be diagonal relations \( ([b], [b], \delta_{[b]}) \), we need to interpret the constants to form a refined \( \lambda_\varepsilon \) model on \( \text{LogRel} \). By the fibration’s faithfulness, this interpretation is merely a property, and not structure we need to provide. Using an inductive argument, ground types \( G \) denote diagonal relations, and if \( p(\text{LogRel}[c]) \) 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:

\[
p(\text{LogRel}[\Gamma \vdash_\varepsilon M : A]) = ([\Gamma \vdash_\varepsilon M : A], [\Gamma \vdash M : A])
\]
We use LogRel to compare the refined model we constructed with the original unrefined model. First:

**Lemma 4.11** Suppose that the free ε-monad \( S_\varepsilon \) is given by the transfinite construction from §2.3. For each morphism \((f_1, f_2) : (X, X) \to (T, T)\) in \( \mathbb{C} \times \mathbb{C} \), if \((f_1, f_2) : (X, X, δ_X) \to (T, T, δ_T)\) then \( f_2 = m^m_\varepsilon \circ f_1 \).

We can now show that the refined semantics is complete for equational reasoning.

**Theorem 4.12 (Completeness)** Under the combined assumptions of this subsection, for all contexts \( \Gamma \) containing only ground types and terms \( \Gamma \vdash M : G \) and \( \Gamma \vdash N : G \) of ground type,

\[
[\Gamma \vdash M : G] = [\Gamma \vdash N : G] \iff [\Gamma \vdash ε M : G] = [\Gamma \vdash ε N : G]
\]

**Proof.** Noting that ground types are interpreted as diagonal relations, we apply Lemma 4.11 to both LogRel\([M]\) and LogRel\([N]\) to show that

\[
[\Gamma \vdash M : G] = m^m_\varepsilon \circ [\Gamma \vdash ε M : G] \quad [\Gamma \vdash N : G] = m^m_\varepsilon \circ [\Gamma \vdash ε N : G]
\]

Now the result follows from the fact that every \( M \)-morphism is a monomorphism. \(\square\)

## 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:

\[
T_\emptyset = \text{Id} \quad T_{(\text{get})} = S \Rightarrow (-) \quad T_{(\text{set})} = (1 + \mathbb{Z})^k \times (-) \quad T_{(\text{get,set})} = T
\]

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

**Example 5.2** If instead of \( T \) we use the monad \( T' = (S \Rightarrow (-) \Rightarrow R) \Rightarrow S \Rightarrow R \), 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 Lawvere 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 \( \omega \text{Cpo} \), we can re-cast Kammar and Plotkin’s [11] validation of effect-dependent optimisations.

**Example 5.4** Let \( \text{value} \) be a base type for values (with associated constants). Consider \( λ_\varepsilon \) with a base type ref of references, and the set \( \Sigma = \{ \text{lookup} : \text{ref} \to \text{value}, \text{update} : \text{ref} \times \text{value} \to \text{value}, \text{alloc} : \text{value} \to \text{ref} \} \) of operations, so that we can read from and write to references, and allocate new references. Let \( I \) be the category of finite ordinals and injections between them. Plotkin and Power [27] interpret these operations the functor category \([I, \text{Set}]\) as follows. Let \( \mathbb{V} \) be a nonempty finite set of values with interpretation for the value constants. Then we interpret \( \text{value} \) as the constantly-\( \mathbb{V} \) functor, and ref as the Yoneda embedding \([\text{ref}] = I(1, -)\), so that \([\text{ref}]\) \( n \) has \( n \) elements. The local state monad is defined using a coend:

\[
TXn := \mathbb{V}^n \Rightarrow \int_{m \in \mathbb{N}} \mathbb{V}(n, m) \times \mathbb{V}^m \times Xm
\]

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 \( n \leq m \)), a new state in \( \mathbb{V}^m \), and a result in \( Xm \).

The category \([I, \text{Set}]\) 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 \( S_\alpha \) converges at \( \mathbb{N} \). We can therefore show by induction on \( \alpha \) that, for example, there are component-wise surjections from the corresponding free monads into the following functors:

\[
T_{\text{(alloc)}} Xn := \int_{m \in \mathbb{N}} I(n, m) \times \mathbb{V}^{m-n} \times Xm \quad T_{\text{(update,lookup)}} Xn := \mathbb{V}^n \Rightarrow \mathbb{V}^n \times Xn
\]

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. For an example of reasoning using
this factorisation, note that there are two sequencing morphisms \( T_{\text{alloc}}(X \times Y) \rightarrow T_{\text{alloc}}(X \times Y) \), one that does the left computation first and one that does right first. It is easy to check that these are equal, i.e., \( T_{\text{alloc}} \) 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 [30] 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 [28,1], 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 \( \omega \text{Cpo} \) 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 will carry over to such settings.

Acknowledgements. This work has been supported by an Engineering and Physical Sciences Research Council (EPSRC) 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, Ian Orton, and Philip Saville for fruitful discussions and suggestions.

References


