Parametric Effect Monads and Semantics of Effect Systems

Shin-ya Katsumata
Research Institute for Mathematical Sciences, Kyoto University, Kyoto, Japan
sinya@kurims.kyoto-u.ac.jp

Abstract

We study fundamental properties of a generalisation of monad called *parametric effect monad*, and apply it to the interpretation of general effect systems whose effects have sequential composition operators. We show that parametric effect monads admit analogues of the structures and concepts that exist for monads, such as Kleisli triples, the state monad and the continuation monad, Plotkin and Power's algebraic operations, and the categorical $\top\top$-lifting. We also show a systematic method to generate both effects and a parametric effect monad from a monad morphism. Finally, we introduce two effect systems with explicit and implicit subeffecting, and discuss their denotational semantics and the soundness of effect systems.

Categories and Subject Descriptors D.3.1 [*Programming Languages*]: Formal Definitions and Theory—Semantics; F.3.3 [*Logics and Meanings of Programs*]: Studies of Program Constructs—Type structure

Keywords algebrac operation; computational effect; effect system; lax monoidal functor; monad; parametric effect monad

1. Introduction

Effect system is a type-based approach to statically estimate computational effects caused by programs. The basic idea of effect system is to add to each typing judgement $\Gamma \vdash M : \tau$ an expression e that estimates the scope of M's computational effect. The expression e is called *effect*. The definition of effects depends on the computational effect that the programming language supports, and also the property we would like to know about the computational effect. For instance,

- Various effect systems and their semantics are studied for the analysis of memory usage during the execution of programs [6, 7, 22, 32, 34, 37]. Effects are defined to be sets of atoms $rd_\rho, wr_\rho, init_\rho$ tagged with regions.
- In [27], an effect system for concurrent ML (an extension of ML with communication primitives) is designed to analyse the communication behaviour of programs. There, effects are expressions of a process calculus.
- In [13], Kammar and Plotkin designs an effect system that estimates the set of algebraic operations invoked during the execution of programs.

Statically estimating properties of computational effects brings various benefits to the analysis and transformation of programs. The main purpose of the pioneering work by Lucassen and Gifford [22] was to design the effect system that assists safe scheduling of expressions in parallel computing. Since then, many effect systems have been designed for analysing the behaviour of programs. For instance, control flow analysis can be concisely captured as an effect system [28]. The region-based memory management in functional languages is developed based on the idea of effect system [35]. A series of papers [6–8, 34] study a semantic foundation for some aggressive program transformations that depend on the result of effect analysis. Kammar and Plotkin gave a general algebraic account of the effect-dependent optimisations based on the property of algebraic theories [13].

In the seminal paper [37, 38], Wadler integrated the effect annotation and Moggi's monadic interpretation of computational effects into *effect-annotated monadic type* $T(e, \tau)$. He then posed a question: "What is the denotational semantics of the effect-annotated monadic type?" The main theme of this paper is to propose an answer to this problem.

Semantic Structures for Effect-Annotated Monadic Types An answer to Wadler's question consists of 1) a formulation of the concept of effect, and 2) a mathematical structure corresponding to the effect-annotated monadic type.

One type of solution formulates effects as *join semilattices*, and use the join operator to compute the effect of let expressions. In this solution, each effect represents a set of *events*, and an effect-annotated monadic type $T(e, \tau)$ is given to the programs that cause the computational effects (regarded as events) in e. The typing rule for let expressions is:

$$\frac{\Gamma \vdash M : T(e, \tau) \quad \Gamma, x : \tau \vdash N : T(e', \sigma)}{\Gamma \vdash \mathsf{let}\ x\ \mathsf{be}\ M\ \mathsf{in}\ N : T(e \vee e', \sigma).} \tag{1}$$

This type of solutions captures well the feature of the effect systems for memory usage analysis in [6, 7, 22, 32, 34, 37]. One of the earliest solutions of this type is Filliâtre's *generalised monad* [10].

However, using the join operator to compute the effect of let expressions is not always satisfactory. Here are some examples:

1. Let us consider an effect system that estimates exceptions raised by programs (we ignore exception handlers). We define an effect to be a set of exceptions. Then effects form a join semilattice by the set union. An instance of the typing rule (1) is the following derivation:

$$\frac{\Gamma \vdash \mathsf{raise}^{E_1}_{\tau} : T(\{E_1\}, \tau) \quad \Gamma, x : \tau \vdash \mathsf{raise}^{E_2}_{\sigma} : T(\{E_2\}, \sigma)}{\Gamma \vdash \mathsf{let}\ x\ \mathsf{be}\ \mathsf{raise}^{E_1}_{\tau}\ \mathsf{in}\ \mathsf{raise}^{E_2}_{\sigma} : T(\{E_1\} \cup \{E_2\}, \sigma)}$$

Permission to make digital or hard copies of all or part of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. Copyrights for components of this work owned by others than the author(s) must be honored. Abstracting with credit is permitted. To copy otherwise, or republish, to post on servers or to redistribute to lists, requires prior specific permission and/or a fee. Request permissions from permissions@acm.org.

POPL '14, January 22–24, 2014, San Diego, CA, USA.
Copyright is held by the owner/author(s). Publication rights licensed to ACM.
ACM 978-1-4503-2544-8/14/01…$15.00.
http://dx.doi.org/10.1145/2535838.2535846

RIGHTSLINK

The union of exceptions in the second line is sound, but not precise, because E_2 will never be raised.

2. Let us consider an effect system that estimates output strings of programs. A natural definition of an effect is thus a set of strings. Although effects form a join semilattice by the set union, it is inadequate to use the join operator in the typing rule (1) to compute the output strings of let expressions. Rather, we should use the language concatenation $\{l@l' \mid l \in e, l' \in e'\}$.

Problem 1.1 To compute the effect of let expressions, what kind of a structure shall we assume on effects ?

Problem 1.2 When the structure on effects is given in the answer to Problem 1.1, what is an appropriate semantic structure for the effect-annotated monadic type?

Problem 1.3 How do we construct the structures proposed as a solution to Problem 1.2?

Algebraic Operations with Effects The operations that manipulate computational effects, such as raise for exception raising, read / write for store access, etc., are important ingredients to make programming languages rich and expressive. To represent such operations in the monadic semantics of programming languages, Plotkin and Power introduced *algebraic operations* [30].

Problem 1.4 How do we extend algebraic operations to the semantic structure given as an answer to Problem 1.2?

The Meaning of Delimiting Monads by Effects Wadler pointed out the essence of effects as follows: "*effects delimit the scope of computational effects*" [37, 38]. [1] To elaborate this, suppose that we have a programming language L with computational effects, and a denotational semantics of L using a monad T_0. An effect system designed for L introduces an effect-annotated monadic type $T(e, \tau)$, and it is given to the terms whose computational effects is within e. This suggests that we may see $T(e, \tau)$ as a *predicate* on $T_0[\![\tau]\!]$.

Problem 1.5 How do we denotationally formulate that "effects delimit the scope of computational effects"?

Another issue is the *soundness of effect system*. When the effect system asserts that a program M of a base type b has an effect e, then we expect that the execution of M causes the computational effect within the scope denoted by e. We formulate this as follows. We first specify the scope of computational effects of an effect e by a predicate $S_b e$ on $T_0[\![b]\!]$. Then we consider whether for all judgements $M : T(e, b)$ in the effect system, we have $[\![M]\!] \in S_b e$. If this holds, then we say that the effect system is *sound* with respect to the specifications of effects.

Problem 1.6 How do we establish the soundness of effect systems with respect to given specifications of effects?

1.1 Contributions of this Paper

This paper proposes the following solutions to the above problems. Especially, solutions to Problem 1.3-1.6 are the technical contributions of this paper.

Solution to Problem 1.1 We postulate that effects form a *preordered monoid*. The monoid structure is to compute the effect of let expressions, and the preorder is to compare the scope of effects. The typing rule (1) for let expressions is refined to

$$\frac{\Gamma \vdash M : T(e, \tau) \quad \Gamma, x : \tau \vdash N : T(e', \sigma)}{\Gamma \vdash \mathsf{let}\ x\ \mathsf{be}\ M\ \mathsf{in}\ N : T(e \cdot e', \sigma)}. \tag{2}$$

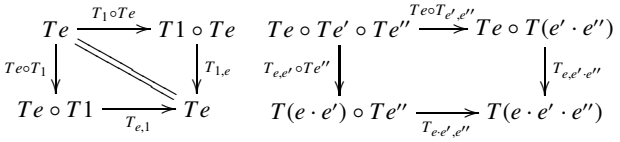
Actually, this typing rule had already appeared in [3] to sketch the idea of effect system, but its semantic account was not given there.

[1] In this paper, the word "delimit" is irrelevant to the *delimited continuation*.

Solution to Problem 1.2 Corresponding to the above solution, for a preordered monoid $\mathbb{E} = (E, \lesssim, 1, \cdot)$ of effects, we employ the following triple (plus a strength, which we omit here):

$$T : \mathbb{E} \to [\mathbb{C}, \mathbb{C}], \quad T_1 : \mathrm{Id} \to T1, \quad T_{e,e'} : Te \circ Te' \to T(e \cdot e')$$

satisfying the axioms of *lax monoidal functor* as a semantic structure for interpreting the effect-annotated monadic type. The morphisms T_1 and $T_{e,e'}$ replace the unit and multiplication of monad. The axioms of monad become the following commuting diagrams:



Such a structure is called a *parametric monad* by Melliès [25], and is introduced as an underlying categorical structure for establishing a bridge between linear logic and the theory of strong monads. In this paper, we consider it as a direct counterpart of the effect-annotated monadic type, and study it from the viewpoint of effect system. As we focus on the parametric monads whose parameter categories are preordered monoids of effects, we call them *parametric effect monads* to emphasise the restriction.

We show that there are parametric analogues of the concepts that exist in the theory of monad, such as Kleisli triples, the state monad, the continuation monad, algebraic operations [30], and the categorical $\top\top$-lifting [14]. This situation suggests that parametric effect monads are a natural generalisation of monads. Moreover, preordered monoids and lax monoidal functors are standard concepts in the theory of monoidal category. Therefore, the tools and results on monoidal categories are available for them (e.g. [24, 25]).

We mention the recent work [33] by Tate, where he introduces respectively *effectors* and *productors* as a solution to Problem 1.1 and 1.2. They are more expressive than parametric effect monads (see Section 6.1). Yet, the aforementioned features of parametric effect monad shows that they have rich structures to be studied.

Solution to Problem 1.3 We give a systematic method to construct *both* a preordered monoid $\mathbb{E}$ of effects and a parametric $\mathbb{E}$-monad on **Set** from a monad morphism $\alpha : \mathsf{T} \to (\mathsf{S}, \sqsubseteq)$; its codomain is a *preordered monad* [16]. We then extend this construction to domain-theoretic setting. Compared to giving effects and parametric effect monads by hand, it is relatively easy to give effect observations. We demonstrate this method with the case where T is the writer monad, exception monad, free algebra monad and probabilistic writer monad, deriving parametric effect versions of these monads.

Solution to Problem 1.4 We extend algebraic operations to parametric effect monads in two ways. One way is to add to the arity of an algebraic operation an effect e that describes its computational effect. An extended algebraic operation has a triple (I, J, e) as its arity, and it is a natural transformation:

$$\alpha_{e',K} : J \Rightarrow Te'K \to I \Rightarrow T(e \cdot e', K)$$

satisfying certain equational axioms. It bijectively corresponds to a morphism of type $I \to TeJ$.

However, when using an algebraic operation of the above type, we have to align the effect of all the arguments to e'. This constraint decreases the accuracy of the estimation of effects in some situations. To remedy this problem, we introduce another way to extend algebraic operations with effects. We allow algebraic operations to have different effects in their argument positions, and describe their effect by an *effect function* ϵ. Then an algebraic operation extended in this way has a tuple (n, ϵ) as its arity, and it is a certain natural

RIGHTSLINK

transformation:

$$\alpha_{e_1,\cdots,e_n,I} : Te_1I \times \cdots \times Te_nI \to T(\epsilon(e_1,\cdots,e_n))I.$$

Solution to Problem 1.5 Our solution takes two steps. First, we introduce the effect system EFi that is designed as a *refinement type system* over Moggi's computational metalanguage (λ_{ML} for short). Each EFi-type τ refines its effect erasure $|\tau|$, which is always a λ_{ML}-type. We next give a denotational semantics of EFi in a *faithful functor* $p : \mathbb{P} \to \mathbb{C}$ with certain structures. We regard $\mathbb{P}$ as a category of predicates on objects in $\mathbb{C}$, and an object X in $\mathbb{P}$ as a *predicate* on the object pX in $\mathbb{C}$. We then interpret each refinement type τ as an object $\langle\tau\rangle$ in $\mathbb{P}$ and its effect erasure as an object $[\![|\tau|]\!]$ in $\mathbb{C}$, so that we have $p\langle\tau\rangle = [\![|\tau|]\!]$, that is, the denotation of a refinement type gives a predicate on the denotation of its erasure. To express that effects delimit the computational effects modelled by a monad T on $\mathbb{C}$, we interpret effect-annotated types by an parametric effect monad $\dot{T}$ on $\mathbb{P}$ such that $p(\dot{T}(e,X)) = \mathrm{T}(pX)$.

Solution to Problem 1.6 After categorically formulating the effect soundness, we give a sufficient conditions for the effect system EFi to be sound with respect to a given specification of effects. This sufficient condition is natural and applicable to any monad, preordered monoid of effects, effect specification and algebraic operation. The proof employs an parametric effect version of *categorical* $\top\top$*-lifting* [14].

Preliminaries

Notation We write $l@l'$ for the concatenation of two sequences l and l'. We write $\mathbf{1}$ for 1) the one-point set $\{*\}$, 2) a terminal object in a category, and 3) a trivial category whose unique object is $*$. The disjoint union of sets $I_1,\cdots,I_n$ tagged with $l_1,\cdots,l_n$ is denoted by $l_1(I_1) + \cdots + l_n(I_n)$.

Category Theory A *thin* category is the one whose homsets have cardinality at most 1. Thin small categories are exactly preordered sets. In this paper, by a bi-CCC we mean a category with chosen finite products, finite coproducts and exponentials.

We write ω**CPO** for the category of all ω-complete posets (which may not contain the least element) and continuous functions between them. We also write **Pre** for the cartesian monoidal category of preordered sets and monotone functions between them.

We use $\bullet$ for the vertical composition and $*$ for the horizontal composition of natural transformations; see [23, Section XII-3] for these operations. For a natural transformation α and a functor F, we write $\alpha \circ F$ for $\alpha * \mathrm{id}_F$ and $F \circ \alpha$ for $\mathrm{id}_F * \alpha$, respectively.

Monad We use sans-serif capital letters $\mathsf{T},\mathsf{S},\cdots$ to denote monads. Its functor part is referred by Roman capital letters $\mathrm{T},\mathrm{S},\cdots$. For a monad $\mathsf{T} = (\mathrm{T},\eta^{\mathsf{T}},\mu^{\mathsf{T}})$ and a morphism $f : I \to \mathrm{T}J$, by $f^{\#_{\mathsf{T}}}$ (or $f^{\#}$ if T is obvious from the context) we mean the *Kleisli lifting* $\mu^{\mathsf{T}}_J \circ \mathrm{T}f$ of f. The monads used in this paper are defined in Table 1.

Computational Metalanguage In this paper, by *computational metalanguage* (λ_{ML} for short) we mean the simply typed lambda calculus with products, coproducts, monadic types and algebraic operations; see [26, 30]. The set $\mathrm{Typ}_{\mathrm{ML}}(B)$ of λ_{ML}-types generated from a set B is defined by the following BNF:

$$\tau ::= b \mid \prod(\tau,\cdots,\tau) \mid \coprod(\tau,\cdots,\tau) \mid \tau \Rightarrow \tau \mid T\tau \quad (b \in B).$$

We specify a λ_{ML} by a λ_{ML}*-signature* $\Delta = (B, O, s)$, where B and O are sets of base types and operator symbols respectively, and $s : O \to H(\mathrm{GTyp}(B)^2)+I(\mathbf{N})$ is a function assigning an arity to each operator symbol $o \in O$; here $\mathrm{GTyp}(B)$ is the subset of $\mathrm{Typ}_{\mathrm{ML}}(B)$ consisting only of base types, product types and coproduct types. See Section 4 for the distinction on arities. A semantics of $\lambda_{\mathrm{ML}}(\Delta)$ is specified by a $\lambda_{\mathrm{ML}}(\Delta)$-structure: a tuple $(\mathbb{C},\mathsf{T},[\![-]\!])$ where $\mathbb{C}$ is a bi-CCC, T is a strong monad on $\mathbb{C}$ and $[\![-]\!]$ assigns:

Category	Symbol	Definition of functor (object part)
Powerset monad / non-empty powerset monad		
Set	P/P^+	$\mathrm{P}I = 2^I$, $\mathrm{P}^+I = 2^I\backslash\{\emptyset\}$
Writer monad		
Set	$\mathsf{Wr}(\Sigma)$	$\mathrm{Wr}(\Sigma)(I) = \Sigma^* \times I$
Exception monad		
Set	$\mathsf{Ex}(E)$	$\mathrm{Ex}(E)(I) = Er(E) + Ok(I)$
Distribution monad		
Set	Ds	$\mathrm{Ds}\,I = \{f : I \to [0,1] \mid \mathrm{supp}(f) : \text{finite}, \sum_{i\in I} f(i) = 1\}$
Lifting monad		
ω**CPO**	L	$\mathrm{L}I = I_\perp$

Table 1. Definition of Monads

1. an object $[\![b]\!] \in \mathbb{C}$ to each $b \in B$ (we extend this to $\mathrm{GTyp}(B)$ using the bi-cartesian structure of $\mathbb{C}$),
2. a $([\![\beta]\!],[\![\beta']\!])$-ary algebraic operation $[\![o]\!]$ for T to each $o \in O$ such that $s(o) = H(\beta,\beta')$, and
3. an n-ary algebraic operation $[\![o]\!]$ for T to each $o \in O$ such that $s(o) = I(n)$.

Monoidal category A monoidal category consists of a category $\mathbb{C}$, an object $\mathbf{I} \in \mathbb{C}$ called the *tensor unit*, a bifunctor $\otimes \in \mathbb{C}^2 \to \mathbb{C}$ called the *tensor product*, and natural isomorphisms l, r, a satisfying coherence axioms; see [23, Section XI-1]. A monoidal category is *strict* if l, r, a are all identities. For every category $\mathbb{C}$, the functor category $[\mathbb{C},\mathbb{C}]$ together with the identity functor $\mathrm{Id}_{\mathbb{C}}$ and the functor composition forms a *strict monoidal category* [23, Exercise XI-3.1]. We use this strict monoidal structure as the default one on $[\mathbb{C},\mathbb{C}]$.

A *lax monoidal functor* ("lax" is dropped in [23, Section XI-2]) between monoidal categories $\mathbb{C},\mathbb{D}$ consists of a functor $F : \mathbb{C} \to \mathbb{D}$, a morphism $F_{\mathbf{I}} : \mathbf{I} \to F\mathbf{I}$ and a natural transformation $F_{I,J} : FI \otimes FJ \to F(I \otimes J)$ satisfying certain conditions. When $F_{\mathbf{I}}$ and $F_{I,J}$ are identities, we replace "lax" with "strict". We write **LaxMonCAT** (resp. **StrictMonCAT**) for the super-large category of strict monoidal categories and lax (resp. strict) monoidal functors between them.

2. Parametric Effect Monads

2.1 Preordered Monoids as Effects

The role of effects in various effect systems is to represent scopes of computational effects caused by programs. As we stated in Section 1, we adopt the following abstract formalisation of the concept of effect.

Postulate 2.1 Effects form a *preordered monoid*.

A preordered monoid is exactly a monoid object in the cartesian monoidal category **Pre**; it consists of a preorder $(E,\lesssim)$, an element $1 \in E$ and a monotone function $(\cdot) : (E,\lesssim)^2 \to (E,\lesssim)$ that satisfy the axioms of monoid. Below, by $e \sim e'$ we mean $(e \lesssim e') \wedge (e' \lesssim e)$. To save space, by the juxtaposition of e and e' we mean $e \cdot e'$. A *partially ordered monoid* is a preordered monoid such that $e \sim e' \iff e = e'$. By *join semilattice* we mean a partially ordered monoid whose monoid structure is given by the join operator. We identify preordered monoids and strict monoidal thin small categories.

2.2 Parametric Effect Monads

We introduce the main subject of this paper, *parametric effect monad*.

RIGHTSLINK

Definition 2.2 Let $\mathbb{E}$ be a preordered monoid. A *parametric* $\mathbb{E}$*-monad* on a category $\mathbb{C}$ is a lax monoidal functor

$$T : \mathbb{E} \to [\mathbb{C}, \mathbb{C}].$$

This is expanded to the following elementary definition:

Definition 2.2-bis Let $\mathbb{E} = (E, \lesssim, 1, \cdot)$ be a preordered monoid. A parametric $\mathbb{E}$-monad consists of the following data.

1. An endofunctor $Te : \mathbb{C} \to \mathbb{C}$ for every $e \in E$.
2. A natural transformation $T(e \lesssim e') : Te \to Te'$ for every $e \lesssim e'$. This satisfies:
$$T(e \lesssim e) = \mathrm{id}_{Te}, \quad T(e' \lesssim e'') \bullet T(e \lesssim e') = T(e \lesssim e'').$$
3. A natural transformation $T_1 : \mathrm{Id} \to T1$.
4. A natural transformation $T_{e,e'} : Te \circ Te' \to T(ee')$ for every $e, e' \in E$.

These data make the following diagrams commute:

$$\begin{array}{ccc} Te_1 \circ Te_1' & \xrightarrow{T_{e_1,e_1'}} & T(e_1e_1') \\ {\scriptstyle T(e_1 \lesssim e_2) * T(e_1' \lesssim e_2')}\downarrow & & \downarrow{\scriptstyle T(e_1e_1' \lesssim e_2e_2')} \\ Te_2 \circ Te_2' & \xrightarrow[T_{e_2,e_2'}]{} & T(e_2e_2') \end{array}$$

$$\begin{array}{ccc} Te & \xrightarrow{T_1 \circ Te} & T1 \circ Te \\ {\scriptstyle Te \circ T_1}\downarrow & \searrow & \downarrow{\scriptstyle T_{1,e}} \\ Te \circ T1 & \xrightarrow[T_{e,1}]{} & Te \end{array} \qquad \begin{array}{ccc} Te \circ Te' \circ Te'' & \xrightarrow{Te \circ T_{e',e''}} & Te \circ T(e'e'') \\ {\scriptstyle T_{e,e'} \circ Te''}\downarrow & & \downarrow{\scriptstyle T_{e,e'e''}} \\ T(ee') \circ Te'' & \xrightarrow[T_{ee',e''}]{} & T(ee'e'') \end{array}$$

Parametric effect monads can be equivalently presented in the form of Kleisli triple.

Definition 2.3 Let $\mathbb{E} = (E, \lesssim, 1, \cdot)$ be a preordered monoid. An *parametric* $\mathbb{E}$*-Kleisli triple* on a category $\mathbb{C}$ consists of the following data.

1. A functor $T - I : \mathbb{E} \to \mathbb{C}$ for every $I \in \mathbb{C}$.
2. A morphism $\eta_I : I \to T1I$ for every $I \in \mathbb{C}$.
3. A mapping $(-)^{e\#e'} : \mathbb{C}(I, Te'J) \to \mathbb{C}(TeI, T(ee')J)$ for every $I, J \in \mathbb{C}$ and $e, e' \in E$. We call this mapping *Kleisli lifting*.

These data satisfy (below $f : I \to TeJ$ and $g : J \to Te'K$)

$$\begin{array}{rcl} (T(e \lesssim e'')J \circ f)^{e'\#e''} & = & T(e'e \lesssim e'e'')J \circ f^{e'\#e} \\ f^{e'\#e} \circ T(e'' \lesssim e')I & = & T(e''e \lesssim e'e)J \circ f^{e''\#e} \\ f^{1\#e} \circ \eta_I & = & f \\ (\eta_I)^{e\#1} & = & \mathrm{id}_{TeI} \\ (g^{e\#e'} \circ f)^{e''\#(ee')} & = & g^{(e''e)\#e'} \circ f^{e''\#e}. \end{array}$$

Proposition 2.4 *Let $\mathbb{E}$ be a preordered monoid. Then parametric $\mathbb{E}$-monads on a category $\mathbb{C}$ bijectively correspond to parametric $\mathbb{E}$ Kleisli triples on $\mathbb{C}$.*

Parametric effect monads are yet insufficient to interpret the typing rule (2). We need *tensorial strengths* on them so that we can add an extra parameter to the Kleisli lifting:

$$(-)^{e\#e'} : \mathbb{C}(I \times J, TeK) \to \mathbb{C}(I \times Te'J, T(e'e)K).$$

We derive the concept of strong parametric effect monad as follows. For a category $\mathbb{C}$ with finite products, we write $[\mathbb{C}, \mathbb{C}]_s$ for the category of *strong endofunctors* and *strong natural transformations* between them [18]. We then equip it with the strict monoidal structure given by the identity functor and the composition of strong endofunctors.

Definition 2.5 Let $\mathbb{C}$ be category with finite products and $\mathbb{E}$ be a preordered monoid. A *strong parametric* $\mathbb{E}$*-monad* is a lax monoidal functor

$$T : \mathbb{E} \to [\mathbb{C}, \mathbb{C}]_s.$$

We can alternatively define the tensorial strength as a *commutator* between a parametric effect monad T and the action of $\mathbb{C}$ to itself; see [25, Section 4]. We note that $[\mathbf{Set}, \mathbf{Set}]_s$ is isomorphic to $[\mathbf{Set}, \mathbf{Set}]$, thus there is no difference between parametric $\mathbb{E}$-monads and strong ones on $\mathbf{Set}$.

2.3 Maps from Parametric Effect Monads to Monads

We introduce a particular type of morphism for parametric effect monads. This will be used in Section 5.2 to express the situation that a parametric effect monad specifies predicates on the underlying monad.f

Definition 2.6 Let $p : \mathbb{C} \to \mathbb{D}$ be a functor, T be a parametric effect monad on $\mathbb{C}$ and $\mathsf{S} = (\mathrm{S}, \eta, \mu)$ be a monad on $\mathbb{D}$. We say that *p maps T to S* if

$$p(TeX) = \mathrm{S}(pX), \quad p((T_1)_X) = \eta_{pX}, \quad p((T_{e,e'})_X) = \mu_{pX}.$$

Though we omit the detail, there are other variations of morphism for parametric effect monads. They are analogues of the morphisms for monads studied in e.g. [19].

2.4 Examples of Parametric Effect Monads

Example 2.7 (Monads) Parametric effect monads subsume monads. Let us write $\mathbf{1}$ for the trivial, one-point preordered monoid. Then parametric $\mathbf{1}$-monads on a category $\mathbb{C}$ bijectively correspond to monads on $\mathbb{C}$. This is an instance of the fact that monoids in a monoidal category $\mathbb{V}$ bijectively correspond to lax monoidal functors of type $\mathbf{1} \to \mathbb{V}$.

Example 2.8 (Parametric Writer Monad) This example shows the writer monad whose output strings are delimited by a given language. Let Σ be a set of alphabets. We consider the following partially ordered monoid $\mathbb{E}$ of languages over Σ:

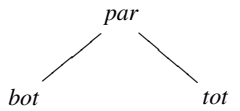
$$\mathbb{E} = (\mathrm{P}(\Sigma^*), \subseteq, \{\epsilon\}, \cdot) \quad \text{where} \quad e \cdot e' = \{l@l' \mid l \in e, l' \in e'\}.$$

Then the following data:

$$TeI = e \times I, \quad (T_1)_I(i) = (\epsilon, i), \quad (T_{e,e'})_I(l, (l', i)) = (l@l', i)$$

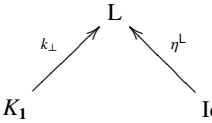
give a parametric $\mathbb{E}$-monad on $\mathbf{Set}$. Although T inherits the structure from the writer monad $\mathsf{Wr}(\Sigma)$, each Te is *not* a monad.

Example 2.9 (Totality Types) This is the semantic analogue of *totality types* by Nielsen et al. [31]. We consider the partially ordered monoid $\mathbb{E} = (\{tot, par, bot\}, \leq, tot, \cdot)$ whose order and multiplication are defined as follows:



$$x \cdot y = \begin{cases} bot & (x = bot \vee y = bot) \\ tot & (x = tot \wedge y = tot) \\ par & (\text{otherwise}) \end{cases}$$

The functor $T : \mathbb{E} \to [\omega\mathbf{CPO}, \omega\mathbf{CPO}]$ mapping the above Hasse diagram to the following diagram in $[\omega\mathbf{CPO}, \omega\mathbf{CPO}]$:



$$\begin{cases} K_1 I & = & \mathbf{1} \\ (k_\perp)_I(*) & = & \perp \end{cases}$$

extends to a parametric $\mathbb{E}$-monad on $\omega\mathbf{CPO}$.

We next give parametric effect versions of the state monad and the continuation monad. For this, we employ a categorical construction

RIGHTSLINK

called *end* [23, Section IX-5]. Let $\mathbb{E} = (E, \lesssim)$ be a preordered set. The *end* of a functor $F : \mathbb{E}^{op} \times \mathbb{E} \to \mathbf{Set}$ is denoted by $\int_{e \in \mathbb{E}} F(e,e)$, and consists of the tuples $\alpha \in \prod_{e \in E} F(e,e)$ such that

$$\forall e, e' \in E \,.\, e \lesssim e' \Rightarrow F(e \lesssim e', e')(\alpha_{e'}) = F(e, e \lesssim e')(\alpha_e).$$

Example 2.10 (Parametric State Monad) Let $\mathbb{E} = (E, \lesssim, 1, \cdot)$ be a preordered monoid and $S : \mathbb{E} \to \mathbf{Set}$ be a (mere) functor. Then the following end

$$TeI = \int_{d \in \mathbb{E}} S\,d \Rightarrow (I \times S\,(de))$$

extends to a parametric $\mathbb{E}$-monad. This end collects the state transformers that take a state of type $S\,d$, and update it to a state of type $S\,(de)$. The effect e abstractly represents what state transformers perform on states. It is added after d because e is the latest action performed on the state.

Example 2.11 (Parametric Continuation Monad) We continue using $\mathbb{E}$ and S in the above example. The following end:

$$TeI = \int_{d \in \mathbb{E}} (I \Rightarrow S\,d) \Rightarrow S\,(ed)$$

extends to a parametric $\mathbb{E}$-monad. An element in TeI is a computation c that takes a continuation k of type $I \Rightarrow S\,d$, and computes a value in the return type $S\,(ed)$. The effect e abstractly represents what c performs on the return type. As continuations are invoked after the main computation of c, d is added after e.

2.5 Monads Indexed by Join-Semilattices

Let $\mathbb{E}$ be a poset. Let us call a functor of type $\mathbb{E} \to \mathbf{Mnd}(\mathbb{C})$ an $\mathbb{E}$-*indexed monad*, where $\mathbf{Mnd}(\mathbb{C})$ is the category of monads on $\mathbb{C}$ and monad morphisms between them. It is employed to model the layered structure of computational effects in Filinski's multi-monadic meta language M3L [9]. The typing rule for let expressions in M3L:

$$\frac{\Gamma \vdash M : T(e,\sigma) \quad \Gamma, x : \sigma \vdash N : T(e',\tau) \quad e \le e'}{\Gamma \vdash \mathsf{let}\ x\ \mathsf{be}\ M\ \mathsf{in}\ N : T(e',\tau)}$$

is reminiscent to the one in general effect system. Indeed, when $\mathbb{E}$ is a join semilattice, the general typing scheme (2) augmented with the side condition $e \le e'$ yields the above rule, because $T(e \vee e', \tau) = T(e', \tau)$. We can say more than this syntactic relationship.

Theorem 2.12 *Let $\mathbb{E}$ be a join semilattice. Then $\mathbb{E}$-indexed monads bijectively correspond to parametric $\mathbb{E}$-monads.*

This theorem is an analogue of [33, Theorem 3], which is stated within effector / productor framework. Filliâtre's *generalized monads* [10] also determine join-semilattice indexed monads that map the least element to the identity monad. There are many examples of monads indexed by join semilattices; see e.g. [13, 33].

3. Parametric Effect Monads via Effect Observations

We next introduce a construction of both a preordered monoid $\mathbb{E}$ and a parametric $\mathbb{E}$-monad on **Set** from *effect observations*. This construction is based on the following nature of effects.

1. *Effects are expressions of an ordered algebra*. This point becomes clear when moving to an elaborated effect system, such as the one for behaviour analysis of concurrent ML programs [27]. There, effects are process algebra expressions ordered by their behaviour.

   We capture an ordered algebra structures on effects by a *preordered monad* $(\mathsf{S}, \sqsubseteq)$ on **Set** [16]. It is a pair of a monad S on **Set** and an assignment $\sqsubseteq$ of a preorder $\sqsubseteq_I$ on $\mathrm{S}I$ to each set I satisfying:

   (substitutivity) for every function $f : I \to \mathrm{S}J$, $f^{\#}$ is a monotone function of type $(\mathrm{S}I, \sqsubseteq_I) \to (\mathrm{S}J, \sqsubseteq_J)$ and

   (congruence) for every function $f, g : I \to \mathrm{S}J$, $f \sqsubseteq_J g$ implies $f^{\#} \sqsubseteq_J g^{\#}$ (here $\sqsubseteq_J$ is extended to the pointwise order for functions).

   The assignment $\sqsubseteq$ bijectively corresponds to a pointwise **Pre**-enrichment on $\mathbf{Set}_{\mathsf{S}}$; see [16] for the detail.

2. *Effects are abstractions of computational effects*. Each effect abstractly expresses some aspects of actual computational effects caused by programs.

   We capture this situation by considering a monad T that models programming language's computational effects, and a *monad morphism* $\alpha : \mathsf{T} \to \mathsf{S}$ that observes computational effects and gives their abstract representations (i.e. effects).

We package the above data into an *effect observation*.

Definition 3.1 An *effect observation* of a monad T on **Set** consists of the following data:

1. A preordered monad $(\mathsf{S}, \sqsubseteq)$ on **Set**.
2. A monad morphism $\alpha : \mathsf{T} \to \mathsf{S}$.

The notation for an effect observation (of T) is $\alpha : \mathsf{T} \to (\mathsf{S}, \sqsubseteq)$.

First, from an effect observation we construct a preordered monoid of effects. Let us write Th_{S} for the algebraic theory corresponding to S. We then define an effect to be a Th_{S}-polynomial in one variable $*$; in other words, an element in $\mathrm{S}\mathbf{1}$. Such polynomials carry the canonical monoid structure given by the variable substitution: the monoid unit 1 is the variable $*$ itself, and the multiplication $\cdot$ is the substitution $e \cdot e' = e[e'/*]$ of e' into $*$ in e. We express this monoid structure in terms of S:

Definition 3.2 Let $(\mathsf{S}, \sqsubseteq)$ be a preordered monad on **Set**. We define the preordered monoid $(\mathrm{S}\mathbf{1}, \sqsubseteq_{\mathbf{1}}, 1, \cdot)$ by

$$1 = \eta^{\mathsf{S}}_{\mathbf{1}}(*), \quad e \cdot e' = (\lambda x \in \mathbf{1} \,.\, e')^{\#S}(e). \tag{3}$$

We denote this by $\mathrm{S}\mathbf{1}$.

Next, we construct a parametric $\mathrm{S}\mathbf{1}$-monad. Let $e \in \mathrm{S}\mathbf{1}$ be an effect. We consider the composite function $v_I = \alpha_{\mathbf{1}} \circ \mathrm{T}!_I : \mathrm{T}I \to \mathrm{S}\mathbf{1}$. It first extracts the computation part of $x \in \mathrm{T}I$ by replacing all the values inside x to $*$ ($\in \mathbf{1}$). Then the monad morphism $\alpha_{\mathbf{1}}$ gives its abstract representation as an effect. We then express that "the computational effect of x is included in the scope of e" by the inequality $v_I(x) \sqsubseteq_{\mathbf{1}} e$, which we use to delimit $\mathrm{T}I$ by effects.

Theorem 3.3 *Let $\alpha : \mathsf{T} \to (\mathsf{S}, \sqsubseteq)$ be an effect observation of a monad T on* **Set**. *We define the following assignment D of sets:*

$$DeI = \{x \in \mathrm{T}I \mid \alpha_{\mathbf{1}} \circ \mathrm{T}!_I(x) \sqsubseteq_{\mathbf{1}} e\} \quad (e \in \mathrm{S}\mathbf{1}, I \in \mathbf{Set})$$

together with the inclusion function $i_I : DeI \subseteq TI$. Then:

1. *The unit η^{T}_I restricts to a function of type $I \to D1I$.*
2. *For all function $f : I \to De'J$, $(i_J \circ f)^{\#\mathsf{T}}$ restricts to a function of type $DeI \to D(ee')J$.*
3. *The tuple of D, η^{T} and $(f)^{e\#e'} = (i \circ f)^{\#\mathsf{T}}$ is a parametric $\mathrm{S}\mathbf{1}$-Kleisli triple.*

When giving an effect system, it is desirable to have the join operator on effects (apart from a monoid structure), because we can use it to unify the effects given to different branches of case expressions. The following examples mainly observe a **Set**-monad T that models computational effects by means of another monad S whose preorder admits the join operator, so that $\mathrm{S}\mathbf{1}$ also admits the

RIGHTSLINK

join operator. In some cases S is given as the extension of T with the join operator.

Example 3.4 We construct a parametric effect monad that is suitable for estimating exceptions raised by programs. There is a distributive law $\mathrm{Ex}(E) \circ \mathrm{P}^+ \to \mathrm{P}^+ \circ \mathrm{Ex}(E)$, and from this we obtain an effect observation of $\mathsf{Ex}(E)$:

$$\eta^{\mathrm{P}^+} \circ \mathrm{Ex}(E) : \mathsf{Ex}(E) \to (\mathsf{P}^+ \circ \mathsf{Ex}(E), \subseteq).$$

Definition 3.2 gives the following partially ordered monoid:

$$(\mathrm{P}^+(\mathrm{Ex}\,\mathbf{1}), \subseteq, \{Ok(*)\}, \cdot),$$

whose monoidal product is given by

$$e \cdot e' = \begin{cases} e & Ok(*) \notin e \\ (e \backslash \{Ok(*)\}) \cup e' & Ok(*) \in e. \end{cases}$$

The parametric $\mathrm{P}^+(\mathrm{Ex}\,\mathbf{1})$-monad D induced by Theorem 3.3 is

$$DeI = \{Er(a) \mid Er(a) \in e\} \cup \{Ok(v) \mid v \in I, Ok(*) \in e\}.$$

We illustrate how effects in $\mathrm{P}^+(\mathrm{Ex}\,\mathbf{1})$ and the parametric effect monad D describes computational effects.

- $c_1 \in D\{Ok(*)\}I$ is a computation that terminates normally, without raising any exception.
- $c_2 \in D\{Er(a), Ok(*)\}I$ is a computation that either terminates normally or may raise an exception a, but no other exception will be raised.
- $c_3 \in D\{Er(b), Er(c)\}I$ is a computation that will raise either an exception b or c, but do nothing else, including normal termination.

The monoid structure of $\mathrm{P}^+(\mathrm{Ex}\,\mathbf{1})$ appropriately captures the fact that the exception raising cancels the rest of the computation. For instance, consider the sequential execution of the computation c_3 followed by c_2. As c_3 never terminates normally, we will not observe the computational effect caused by c_2. The monoid multiplication of $\mathrm{P}^+(\mathrm{Ex}\,\mathbf{1})$ gives the following effect to the sequential execution $c_3; c_2$:

$$\{Er(b), Er(c)\} \cdot \{Er(a), Ok(*)\} = \{Er(b), Er(c)\},$$

which correctly captures the computational effect of $c_3; c_2$.

Example 3.5 Generalising the previous example, let Σ be a first-order single-sorted signature (without equational axioms) and O be the set of symbols defined in Σ. We write T_Σ for the monad of free-Σ algebra, and aim to design the parametric effect monad that estimates the occurrences of operators in $t \in \mathsf{T}_\Sigma I$, and also the closedness of t.

The monad morphism we use to observe T_Σ is the following $|-| : \mathsf{T}_\Sigma \to \mathsf{P}(Op(O) + Var(-))$:

$$|x| = \{Var(x)\}, \qquad |o(t_1, \cdots, t_n)| = \{Op(o)\} \cup |t_1| \cup \cdots \cup |t_n|$$

It collects the operator symbols and variables occurring in t. The partially ordered monoid derived from the effect observation $|-| : \mathsf{T}_\Sigma \to (\mathsf{P}(Op(O) + Var(-)), \subseteq)$ is

$$(\mathsf{P}(Op(O) + Var(\mathbf{1})), \subseteq, \{Var(*)\}, \cdot),$$

whose multiplication is given by

$$e \cdot e' = \begin{cases} e & Var(*) \notin e \\ (e \backslash \{Var(*)\}) \cup e' & Var(*) \in e. \end{cases}$$

The parametric $\mathrm{P}(Op(O) + Var\mathbf{1})$-monad D derived by Theorem 3.3 is

$$DeI = \{t \in T_\Sigma I \mid |t[*/i]| \subseteq e\}.$$

We illustrate how this parametric effect monad estimates occurrences of operator symbols and closedness of Σ-terms.

- $t \in D\{Op(s), Op(z), Var(*)\}I$ is a Σ-term consisting of operator symbols s, z and variables. It may be closed. No other operator occurs in t.
- $t \in D\{Op(c), Op(f)\}I$ is a closed Σ-term consisting of the operator symbols f, c only.

Example 3.6 We derive Example 2.8 by Theorem 3.3. We consider the writer monad $\mathsf{Wr}(\Sigma)$. There is a distributive law $\mathrm{Wr}(\Sigma) \circ \mathrm{P} \to \mathrm{P} \circ \mathrm{Wr}(\Sigma)$, and from this we obtain an effect observation of $\mathsf{Wr}(\Sigma)$:

$$\eta^{\mathrm{P}} \circ \mathrm{Wr}(\Sigma) : \mathsf{Wr}(\Sigma) \to (\mathsf{P} \circ \mathsf{Wr}(\Sigma), \subseteq)$$

Definition 3.2 yields the partially ordered monoid of languages over Σ, and Theorem 3.3 yields the parametric effect monad that is (isomorphic to) the one in Example 2.8.

Example 3.7 We consider modelling the programming language that has a character output operator out_c and a binary probabilistic choice operator $- \oplus_p -$, which chooses the left computation with probability p or the right one with probability $1 - p$. We model these computational effects by the composite monad $\mathsf{Ds} \circ \mathsf{Wr}(\Sigma)$ of the distribution monad and the writer monad.

We first derive a parametric effect monad that estimates output strings of programs, ignoring their output probability. For this, we use the monad morphism $\mathrm{supp} : \mathsf{Ds} \to \mathsf{P}^+$ that computes the *support* of a given distribution:

$$\mathrm{supp}_I(d) = \{i \in I \mid d(i) \neq 0\}.$$

By composing the writer monad, we obtain the following effect observation of $\mathsf{Ds} \circ \mathsf{Wr}(\Sigma)$:

$$\mathrm{supp} \circ \mathrm{Wr}(\Sigma) : \mathsf{Ds} \circ \mathsf{Wr}(\Sigma) \to (\mathsf{P}^+ \circ \mathsf{Wr}(\Sigma), \subseteq).$$

We apply Theorem 3.3 to this situation. It yields the partially ordered monoid of non-empty languages $\mathrm{P}^+(\Sigma^*)$ over Σ (c.f. Example 2.8). The parametric $\mathrm{P}^+(\Sigma^*)$-monad is then given as

$$\begin{aligned} DeI &= \{d \in \mathrm{Ds}(\mathrm{Wr}(\Sigma)(I)) \mid \mathrm{supp}_{\mathrm{Wr}(1)} \circ \mathrm{Ds}(\mathrm{Wr}(\Sigma)(!_I))(d) \subseteq e\} \\ &= \{d \in \mathrm{Ds}(\Sigma^* \times I) \mid \forall l \in \Sigma^*, i \in I \,.\, d(l, i) \neq 0 \implies l \in e\}. \end{aligned}$$

Example 3.8 (Continued from the above example) We next derive another parametric effect monad that takes it into account the probability of output strings. Following the pattern of Example 3.4 and 2.8, we extend the (algebraic theory of) distribution monad Ds with the join operator, then use the embedding of Ds to the extended monad as an effect observation. The extension of Ds with the join operator is known to yield the monad that collects *convex subsets of distributions* [12, 36], which we explain below.

Let $X \subseteq \mathrm{Ds}\, I$ be a subset of probability distributions over I. The *convex closure* of X collects all the probabilistic combinations of distributions in X:

$$CX = \bigcup_{n=0}^{\infty} \left\{ \textstyle\bigoplus_{i=1}^n t(i) \cdot d_i \mid t \in \mathrm{Ds}\{1, \cdots, n\}, d : X^n \right\}.$$

When $CX = X$, we call X *convex*. The monad Cv of *convex subsets of distributions* then collects all the finitely generated non-empty convex subsets of $\mathrm{Ds}\, I$:

$$\mathrm{Cv}\, I = \{CX \mid \emptyset \subsetneq X \subseteq_{\mathrm{fin}} \mathrm{Ds}\, I\}.$$

The inclusion relation between convex subsets gives a preorder on Cv. We write $\boxplus_{p,I} : (\mathrm{Cv}\, I)^2 \to \mathrm{Cv}\, I$ for the probabilistic summation of two convex subsets:

$$X \boxplus_{p,I} Y = \{d \oplus_p d' \mid d \in X, d' \in Y\}.$$

The singleton set function $\{-\}_I : \mathrm{Ds}\, I \to \mathrm{Cv}\, I$ is a monad morphism, hence we obtain an effect observation

$$\{-\} \circ \mathrm{Wr}(\Sigma) : \mathsf{Ds} \circ \mathsf{Wr}(\Sigma) \to (\mathsf{Cv} \circ \mathsf{Wr}(\Sigma), \subseteq).$$

RIGHTSLINK

By applying Theorem 3.3, we obtain the partially ordered monoid $(\mathrm{Cv}(\Sigma^*), \subseteq, \{\{\epsilon : 1\}\}, \cdot)$, and a parametric $\mathrm{Cv}(\Sigma^*)$-monad:

$$DeI = \{d \in \mathrm{Ds}(\Sigma^* \times I) \mid \{\overline{d}\} \in e\},$$

where $\overline{d} \in \mathrm{Ds}(\Sigma^*)$ is the distribution given by $\overline{d}(l) = \sum_{i \in I} d(l, i)$.

We see the role of the convexity in effects. Consider the following conditional expression.

$$M = \text{if } N \text{ then } out_a() \text{ else } out_b().$$

This is a *deterministic* program that outputs either a or b. The summary of the side effect is thus "M outputs a with probability 1 or b with probability 1", but it is yet unclear what is the probability of choosing "then" branch and "else" branch by the conditional expression. The monad Cv handles this situation by taking the convex closure:

$$C(\{\{a : 1\}, \{b : 1\}\}) = \{\{a : p, b : (1 - p)\} \mid p \in [0, 1]\}$$

to give the overall probability of the output strings of M.

3.1 Effect Observations in an Order-Enriched Setting

We next give an analogue of Theorem 3.3 in a **Pre**-enriched category $\mathbb{C}$. We assume that $\mathbb{C}$ has a terminal object $\mathbf{1}$, and that $\mathbb{C}$ has chosen comma objects of the following form:

$$\begin{array}{ccc} f \downarrow j & \xrightarrow{!} & \mathbf{1} \\ {\scriptstyle \pi_{f,j}}\downarrow & \lesssim & \downarrow{\scriptstyle j} \\ I & \xrightarrow[f]{} & J \end{array} \tag{4}$$

An informal way to see the comma object $f \downarrow j$ is that it is the pullback of the inclusion of the downset $\downarrow j \hookrightarrow J$ along f.

Below we write $\mathbb{C}_0$ for the underlying ordinary (i.e. **Set**-enriched) category of $\mathbb{C}$. An *effect observation* of a monad T on $\mathbb{C}_0$ by a **Pre**-enriched monad S on $\mathbb{C}$ is a monad morphism $\alpha : \mathsf{T} \to \mathsf{S}_0$, where S_0 is the underlying ordinary monad on $\mathbb{C}_0$.

Theorem 3.9 *Under the above $\mathbb{C}$ and an effect observation $\alpha : \mathsf{T} \to \mathsf{S}_0$, the following $\mathbb{E}$ is a preordered monoid, and D is a parametric $\mathbb{E}$-monad on $\mathbb{C}_0$.*

$$\begin{array}{rcl} \mathbb{E} & = & (\mathbb{C}(\mathbf{1}, \mathsf{S}\mathbf{1}), \quad \lesssim_{\mathbf{1},\mathsf{S}\mathbf{1}}, \quad \eta^{\mathsf{S}}_{\mathbf{1}}, \quad \lambda f g \,.\, g^{\#_{\mathsf{S}}} \circ f) \\ DeI & = & (\alpha_{\mathbf{1}} \circ \mathsf{T}!_I) \downarrow e \end{array}$$

Example 3.10 We derive a parametric effect monad whose effects can capture the behaviour of non-terminating programs. We consider the computational effect of divergence and bell ringing, and represent it by a monad on **ADCPO**, the category of algebraic DCPOs and all continuous functions between them. This category is **Pos**-enriched, has a terminal object 1 and has the comma object of the form (4). It is also closed under the lower powerdomain construction (see e.g. [1]).

We use the lazy counter monad, which is denoted by B in this example, to represent divergence and bell ringing. The ADCPO $\mathrm{B}I$ is given as the least solution of the recursive domain equation $X \simeq (I + X)_\perp$, and its Hasse diagram is depicted below (in the diagram, i ranges over I, and $x \to y$ means $x \le y$):

$$\mathrm{B}I = \mu Y.(I + Y)_\perp \qquad \begin{array}{ccccccccc} & (0,i) & & (1,i) & & (2,i) & & \cdots & \\ & \uparrow & & \uparrow & & \uparrow & & & \\ 0\perp & \longrightarrow & 1\perp & \longrightarrow & 2\perp & \longrightarrow & \cdots & \longrightarrow & \infty\perp \end{array}$$

Each element in $\mathrm{B}I$ denotes a phenomenon that may happen by executing a program.

- An element $x\perp$ ($x \in \mathbf{N}$) stands for the phenomenon that "the program rings the bell x times and it is still running".
- The element $\infty\perp$ stands for the phenomenon that "$x\perp$ holds for all $x \in \mathbf{N}$". Thus the program rings the bell forever.
- The element (x, i) stands for the phenomenon that "the program rings the bell x times, then it terminates and returns a value i". Especially, when $I = \mathbf{1}$, the element $(x, *)$ stands for that "the program rings the bell x times and terminates".

The order $\le$ on these elements captures the implication relation between corresponding phenomena.

- The order $(x, i) \ge x\perp$ reflects that if a program terminates after ringing the bell x times, then the program *is* indeed running right after the x-th bell.
- The order $x\perp \ge (x - 1)\perp$ reflects that if the phenomenon $x\perp$ happens then $(x - 1)\perp$ also happens before the x-th bell.
- The phenomenon $0\perp$ always happens for any program because there is a silent moment right after the program starts. Thus $0\perp$ is the least element.

We next consider the composite functor $\mathrm{P}_l \circ \mathrm{B}$, where P_l is the lower powerdomain construction. The composite extends to a **Pos**-(i.e. locally monotone) monad over the **Pos**-category **ADCPO**. We then obtain a monad morphism

$$\eta^{\mathrm{P}_l} \circ \mathsf{B} : \mathsf{B} \to \mathrm{P}_l \circ \mathsf{B},$$

to which we can apply Theorem 3.9.

The carrier set of the partially ordered monoid $\mathrm{P}_l(\mathrm{B}\mathbf{1})$ given by Theorem 3.9 is isomorphic to the full-sub poset C of $(\mathbb{N} \cup \{\infty\}, \le) \times (P(\mathbb{N}), \subseteq)$ consisting of pairs (k, X) such that $\bigsqcup X \le k$. Each pair (k, X) encodes the following subset of $\mathrm{B}\mathbf{1}$ (below $\downarrow x$ denotes the downset of x):

$$\Phi(k, X) = \downarrow (k\perp) \cup \bigcup_{x \in X} \downarrow (x, *),$$

and an effect (k, X) means that one of the phenomena in $\Phi(k, X)$ happens. We note that $(k, \emptyset) \in C$ expresses the behaviour of programs that never terminate.

The unit of $\mathbb{E}$ is $(0, \{0\})$, and the multiplication is

$$(k, X) \cdot (l, Y) = \left(k \sqcup \left(\bigsqcup_{x \in X} (x + l)\right), \{x + y \mid x \in X, y \in Y\}\right).$$

We especially have $(k, \emptyset) \cdot (l, Y) = (k, \emptyset)$, corresponding to that when the first expression of the sequential composition never terminates, the second expression will never be executed, thus its effect will be discarded.

The parametric $\mathrm{P}_l(\mathrm{B}\mathbf{1})$-monad by Theorem 3.9 is the following. With the helper function $\varphi : \mathrm{B}I \to \mathrm{P}_l(\mathrm{B}\mathbf{1})$ defined by:

$$\varphi(x, i) = (x, \{x\}), \quad \varphi(x\perp) = (x, \emptyset)$$

the parametric $\mathrm{P}_l(\mathrm{B}\mathbf{1})$-monad D is given by

$$DeI = \{x \in \mathrm{B}I \mid \varphi(x) \in \Phi(e)\}.$$

4. Algebraic Operations for Parametric Effect Monads

We extend Plotkin and Power's *algebraic operations* introduced in [30] to strong parametric effect monads. A straightforward extension is the following:

Definition 4.1 Let $\mathbb{E}$ be a preordered monoid and T be an strong parametric $\mathbb{E}$-monad on a CCC $\mathbb{C}$. For $I, J \in \mathbb{C}$ and $e \in \mathbb{E}$, an (I, J, e)-ary *algebraic operation* for T is a family of morphisms $a_{e',K} : J \Rightarrow Te'K \to I \Rightarrow T(ee')K$, natural on e', such that for

RIGHTSLINK

any $f : L \times K \to Te''M$, the following square commute:

$$\begin{array}{ccccc} L \times (J \Rightarrow Te'K) & \xrightarrow{c} & J \Rightarrow (L \times Te'K) & \xrightarrow{J \Rightarrow f^{\#}} & J \Rightarrow T(e'e'')M \\ \downarrow{\scriptstyle L \times a_{e,K}} & & & & \downarrow{\scriptstyle a_{e,M}} \\ L \times (I \Rightarrow T(ee')K) & \xrightarrow[c]{} & I \Rightarrow (L \times T(ee')K) & \xrightarrow[I \Rightarrow f^{\#}]{} & I \Rightarrow T(ee'e'')M \end{array}$$

For $n \in \mathbf{N}$ and $e \in \mathbb{E}$, an (n, e)-ary algebraic operation is similarly defined by replacing $J \Rightarrow -$ with $(-)^n$ and $I \Rightarrow -$ with Id in the above diagram.

An easy calculation shows that (I, J, e)-ary algebraic operations for T bijectively correspond to morphisms of type $I \to TeJ$. This is an analogue of the correspondence between algebraic operations and *generic effects* stated in [30].

4.1 Algebraic Operations with Different Effect Arguments

The above extension of algebraic operations is natural, but not satisfactory in some situations. The reason is twofold: 1) a single effect e may not be precise enough to capture the effect of an algebraic operation, and 2) the arguments of an algebraic operation has to have the same effect.

Let us see these problems with the parametric $\mathrm{Cv}(\Sigma^*)$-monad D over the distribution monad Ds in Example 3.8. We consider restricting the domain and the codomain of $\oplus_p$ to obtain a $(2, e')$-ary algebraic operation for D. Following Definition 4.1, it is a certain family of functions of the following type:

$$\oplus_{p,I} : DeI \times DeI \to D(e'e)I.$$

What is an appropriate choice for e'? Actually $1 = \{\{\epsilon : 1\}\}$ is the the best. Since the choice operator itself does not output any string, if e' contains some output strings with non-zero probability, then it introduces garbage to the estimation of output strings. We then realise that the effect 1 does not describe the behaviour of the computational effect $\oplus_p$.

Even if we accept $e' = 1$, we have another unsatisfactory point in using the algebraic operation of the above type. When supplying two computations $c_1 \in De_1I$ and $c_2 \in De_2I$ having different effects to $\oplus_{p,I}$, we first need to align their effects to, for example, $c_1, c_2 \in D(e_1 \vee e_2)I$. Then the probabilistic choice of them yields the computation $c_1 \oplus_{p,I} c_2 \in D(e_1 \vee e_2)I$, but this is too rough; for instance even when $p = 0$ (i.e. discarding the left argument) the effect of the left computation survives after the probabilistic choice.

Our solution to these problems is to allow the effects in the argument position of algebraic operations to be different with each other. For instance, in the context of Example 3.8, we give the following domain and codomain to $\oplus_p$:

$$(\oplus_p)_{e_1,e_2,I} : De_1I \times De_2I \to D(e_1 \boxplus_{p,\Sigma^*} e_2)I \tag{5}$$

so that the effect can say more precisely how argument effects are processed by the algebraic operation. We formalise this idea below.

Definition 4.2 Let $\mathbb{E} = (E, \lesssim, 1, \cdot)$ be a preordered monoid and T be an strong parametric $\mathbb{E}$-monad on a category $\mathbb{C}$ with finite products.

1. An n-ary *effect function* on $\mathbb{E}$ is a functor $\epsilon : \mathbb{E}^n \to \mathbb{E}$ such that
$$\epsilon(e_1, \cdots, e_n) \cdot e' = \epsilon(e_1e', \cdots, e_ne').$$
Below we abbreviate the sequence $e_1e', \cdots, e_ne'$ to $\vec{e}e'$.
2. Let ϵ be an n-ary effect function on $\mathbb{E}$. An (n, ϵ)-*ary algebraic operation* for T is a family of morphisms
$$a_{e_1,\cdots,e_n,I} : Te_1I \times \cdots \times Te_nI \to T(\epsilon(e_1, \cdots, e_n))I,$$
which is natural on $e_1, \cdots, e_n \in E$, such that for any $f : K \times I \to Te'J$, the following diagram commute:
$$\begin{array}{ccccc} K \times \prod Te_iI & \xrightarrow{\langle K \times \pi_i \rangle_{i=1}^n} & \prod (K \times Te_iI) & \xrightarrow{\prod f^{e_i \# e'}} & \prod T(e_ie')J \\ \downarrow{\scriptstyle K \times a_{\vec{e},I}} & & & & \downarrow{\scriptstyle a_{\vec{e}e',I}} \\ K \times T(\epsilon(\vec{e}))I & \xrightarrow[f^{\epsilon(\vec{e}) \# e'}]{} & T(\epsilon(\vec{e})e')J & = & T(\epsilon(\vec{e}e'))J \end{array}$$
In the above diagram, $f^{e \# e'} : K \times TeI \to T(ee')J$ is the parametrised Kleisli lifting of f using the strength of T.

Currently, it is not clear how to extend the arity in the above definition to arbitrary objects in $\mathbb{C}$. This technical limitation also affects the design of effect systems in Section 5; there, algebraic operations in effect systems are classified into two groups, one corresponding to Definition 4.1 and the other corresponding to the above definition.

We gave a construction of the parametric effect monad D from an effect observation $\alpha : \mathsf{T} \to (\mathsf{S}, \sqsubseteq)$ in Theorem 3.3. We next show that every n-ary algebraic operation a for T can be restricted to an (n, ϵ)-ary algebraic operation for D, where ϵ is constructed from a and α. Recall that the monad morphism α maps the n-ary algebraic operation a for T to the one for S. We denote it by $\overline{\alpha}(a)$.

Theorem 4.3 *Let $\alpha : \mathsf{T} \to (\mathsf{S}, \sqsubseteq)$ be an effect observation of a monad T on* **Set***, and a be an n-ary algebraic operation for T. We write D for the parametric effect monad derived from the effect observation α by Theorem 3.3.*

1. *Function $\overline{\alpha}(a)_{\mathbf{1}} : (\mathbf{S1})^n \to \mathbf{S1}$ is an n-ary effect function on $\mathbf{S1}$.*
2. *Each a_I restricts to a function of the following type:*
$$a_I : De_1I \times \cdots De_nI \to D(\overline{\alpha}(a)_{\mathbf{1}}(e_1, \cdots, e_n))I,$$
and this is an $(n, \overline{\alpha}(a)_{\mathbf{1}})$-ary algebraic operation for D.

Example 4.4 (Continued from Example 3.8) From Theorem 4.3, the binary algebraic operation $\oplus_p$ for $\mathsf{Ds} \circ \mathsf{Wr}$ restricts to an $(2, \boxplus_{p,\Sigma^*})$-ary algebraic operation for the parametric effect monad in Example 3.8.

Example 4.5 Let Σ be a first-order single-sorted signature (without equational axioms) and T_Σ be the monad of free Σ-algebra. There always exists a distributive law $\mathsf{T}_\Sigma \circ \mathsf{P} \to \mathsf{P} \circ \mathsf{T}_\Sigma$, and we obtain the effect observation

$$\eta : \mathsf{T}_\Sigma \to (\mathsf{P} \circ \mathsf{T}_\Sigma, \subseteq).$$

Definition 3.2 applied to this situation yields the preordered monoid $\mathrm{P}(\mathrm{T}_\Sigma(\mathbf{1}))$. The parametric $\mathrm{P}(\mathrm{T}_\Sigma(\mathbf{1}))$-monad D of Theorem 3.3 is

$$DeI = \{t \in \mathrm{T}_\Sigma I \mid t[*/i]_{i \in I} \in e\}.$$

Each term $t \in \mathrm{T}_\Sigma\{1, \cdots, n\}$ gives the following n-ary algebraic operation a_t for T_Σ:

$$(a_t)_I(t_1, .., t_n) = t[t_i/i]_{i=1}^n.$$

Theorem 4.3 associates to this algebraic operation the following effect function on $\mathrm{P}(\mathrm{T}_\Sigma(\mathbf{1}))$:

$$\overline{\alpha}(a_t)_{\mathbf{1}}(e_1, \cdots, e_n) = \{t[t_i/i]_{i=1}^n \mid t_i \in e_i\}$$

and a_t restricts to the following $(n, \overline{\alpha}(a_t)_{\mathbf{1}})$-ary algebraic operation for D:

$$(a_t)_I : De_1I \times \cdots \times De_nI \to D\{t[t_i/i]_{i=1}^n \mid t_i \in e_i\}I.$$

RIGHTSLINK

5. EFe/EFi: Simply-Typed Monomorphic Effect Systems with Effect Subtyping

We introduce two simply typed monomorphic effect systems, EFe and EFi. These two systems differ in handling effect subtyping. Both styles are adopted in many works. For instance,

- The calculus EFe adopts *explicit subeffecting* by effect coercion operators $T^{e \lesssim e'}$. This language is close to the one considered in Fillinski's M3L [9], where the point we can up-cast effect-annotated types is limited to let expressions. Another example of the effect system that adopts explicit subeffecting is [13].
- The calculus EFi adopts *implicit subeffecting*. Subjects of EFi judgements are λ_{ML}-terms, and the subeffecting rule does not change subjects of judgements. Each type of EFi is a *refinement* of its underlying λ_{ML}-type. See e.g. [4, 8].

These systems are specified by an EF-signature. We define the set $\mathrm{GTyp}(B)$ of ground types generated from B by the following BNF:

$$\mathrm{GTyp}(B) \ni \beta ::= b \mid \prod(\beta, \cdots, \beta) \mid \coprod(\beta, \cdots, \beta) \quad (b \in B).$$

We write $\mathbf{1}$ for $\prod()$ and $\overline{n}$ for $\coprod(\overbrace{\mathbf{1}, \cdots, \mathbf{1}}^{n})$.

Definition 5.1 An EF-signature Σ consists of the following data:

1. A preordered monoid $\mathbb{E} = (E, \lesssim, 1, \cdot)$.
2. A set B of base types.
3. A set O of symbols for algebraic operations.
4. A function giving arities to algebraic operation symbols:

$$\begin{array}{rcl} s : O & \to & H(\mathrm{GTyp}(B)^2 \times E) \\ & + & I\{(n, \epsilon) \mid n \in \mathbf{N}, \epsilon : n\text{-ary effect function on } \mathbb{E}\}. \end{array}$$

The arity $H(\beta, \beta', e)$ and $I(n, \epsilon)$ are given to the algebraic operations in the style of Definition 4.1 and 4.2, respectively. Every EF-signature $\Sigma = (\mathbb{E}, B, O, s)$ determines an λ_{ML}-signature $\Sigma_0 = (B, O, s')$ by discarding the effect information from s.

Throughout this section, we use a fixed EF-signature $\Sigma = (\mathbb{E}, B, O, s)$ with $\mathbb{E} = (E, \lesssim, 1, \cdot)$. Both calculi $\mathrm{EFe}(\Sigma)$ and $\mathrm{EFi}(\Sigma)$ share the same set $\mathrm{Typ}_{\mathrm{EF}}(\Sigma)$ of types defined as follows:

$$\begin{array}{rcl} \mathrm{Typ}_{\mathrm{EF}}(\Sigma) \ni \tau & ::= & b \mid \prod(\tau, \cdots, \tau) \mid \coprod(\tau, \cdots, \tau) \mid \tau \Rightarrow \tau \mid T(e, \tau) \\ & & (b \in B, e \in \mathbb{E}). \end{array}$$

We define the *erasure function* $|-| : \mathrm{Typ}_{\mathrm{EF}}(\Sigma) \to \mathrm{Typ}_{\mathrm{ML}}(\Sigma_0)$ by the one that recursively replaces $T(e, \tau)$ by $T|\tau|$. We extend $|-|$ to typing contexts in the canonical way.

5.1 Explicit Subeffecting Calculus $\mathrm{EFe}(\Sigma)$

The calculus $\mathrm{EFe}(\Sigma)$ extends the simply typed lambda calculus with products and coproducts with the following raw terms:

$$[M], \quad \mathsf{let}^{e,e'}\ x^{\tau}\ \mathsf{be}\ M\ \mathsf{in}\ N, \quad T^{e \lesssim e'}\ M, \quad o(M), \quad o(M, \cdots, M)$$

The first two are parametric analogues of pure-computation constructors and let expressions in λ_{ML}. The third one is the (effect) coercion operator. The last two are the syntax for algebraic operations.

The typing rules are displayed in Figure 1. The equational theory of $\mathrm{EFe}(\Sigma)$ extends the $\beta\eta$-equational theory for the simply typed lambda calculus with products and coproducts by the equational axioms displayed in Figure 2. The axioms (6) and (7) guarantee the functoriality of $T(-, \tau)$. The axioms (8)-(12) are a syntax representation of the axioms of parametric effect Kleisli triple (Definition 2.3). The axioms (13) and (14) guarantee that each $o \in O$ behave as an algebraic operation for T. The axiom (13) is for the symbol $o \in O$ such that $s(o) = H(\beta, \beta', e)$, while (14) is for $o \in O$ such that $s(o) = I(n, \epsilon)$.

The semantics of $\mathrm{EFe}(\Sigma)$ is specified by an $\mathrm{EFe}(\Sigma)$-structure.

Definition 5.2 An $\mathrm{EFe}(\Sigma)$-*structure* consists of the following data.

1. A bi-CCC $\mathbb{C}$ and an strong parametric $\mathbb{E}$-monad T on $\mathbb{C}$.
2. An object $[\![b]\!] \in \mathbb{C}$ for each $b \in B$. We extend this assignment of $\mathbb{C}$-objects to base types to ground types $\mathrm{GTyp}(B)$ (see Preliminaries section) using the bi-cartesian structure on $\mathbb{C}$ in the canonical way.
3. A $([\![\beta]\!], [\![\beta']\!], e)$-ary algebraic operation $[\![o]\!]$ for T, for each $o \in O$ such that $s(o) = H(\beta, \beta', e)$.
4. An (n, ϵ)-ary algebraic operation $[\![o]\!]$ for T, for each $o \in O$ such that $s(o) = I(n, \epsilon)$.

Interpretations of $\mathrm{EFe}(\Sigma)$-types and terms are straightforward.

Theorem 5.3 *Let* $(\mathbb{C}, T, [\![-]\!])$ *be an* $\mathrm{EFe}(\Sigma)$-*structure. For every* $\mathrm{EFe}(\Sigma)$-*judgements* $\Gamma \vdash M, N : \tau$*, if* $M = N$ *holds in the equational theory of* $\mathrm{EFe}(\Sigma)$*, then we have* $[\![M]\!] = [\![N]\!]$.

5.2 Implicit Subeffecting Calculus $\mathrm{EFi}(\Sigma)$

The implicit subeffecting calculus $\mathrm{EFi}(\Sigma)$ is designed to be a refinement type system for the computational metalanguage $\lambda_{\mathrm{ML}}(\Sigma_0)$. The subject M of an $\mathrm{EFi}(\Sigma)$-judgement $\Gamma \vdash M : \tau$ is an $\lambda_{\mathrm{ML}}(\Sigma_0)$-term. Thus raw terms do not contain coercion operators $T^{e \lesssim e'}$, and effect annotations on let expressions are removed. Variable binders are also annotated with $\lambda_{\mathrm{ML}}(\Sigma_0)$-types instead of $\mathrm{EFi}(\Sigma)$-types.

Proposition 5.4 *If* $\Gamma \vdash M : \tau$ *then* $|\Gamma| \vdash_{\mathrm{ML}} M : |\tau|$.

We move to the semantics of $\mathrm{EFi}(\Sigma)$. The basic idea of refinement type is that each refinement type τ specifies a certain part, or a *predicate*, of its underlying type $|\tau|$. To model this idea, we employ a categorical setting that provides the concept of predicate on objects in a category.

A simple setting to talk about predicates on objects in a category $\mathbb{C}$ is to consider a *faithful functor* $p : \mathbb{P} \to \mathbb{C}$. We then regard:

- $\mathbb{P}$ as the category of *predicates* on objects in $\mathbb{C}$,
- $pX = I$ as "$X \in \mathbb{P}$ is a predicate over $I \in \mathbb{C}$",
- $p(\dot{f} : X \to Y) = f : I \to J$ as "f-image of X is included by Y", and the morphism $\dot{f}$ as the unique witness of this statement.

Below, for objects $X, Y \in \mathbb{P}$ and a morphism $f : pX \to pY$ in $\mathbb{C}$, by $f : X \dot{\to} Y$ we mean the statement "there exists a unique $\dot{f} : X \to Y$ in $\mathbb{P}$ such that $p\dot{f} = f$". From the above viewpoint, $\mathrm{id}_I : X \dot{\to} Y$ corresponds to "X implies Y" for predicates X, Y over $I \in \mathbb{C}$. Thus the category consisting of objects X such that $pX = I$ and morphisms $\dot{f}$ such that $p(\dot{f}) = \mathrm{id}_I$ may be seen as the preorder of predicates on I. We name this category $\mathbb{P}_I$, and call it the *fibre category* over I.

Besides, there are products, coproducts, arrow types and effect-annotated monadic types on refinement types; we thus assume that $\mathbb{P}$ is a bi-CCC, and that a parametric effect monad $\dot{T}$ is given on $\mathbb{P}$. The introduction and elimination of these type structures are synchronised with those of the underlying λ_{ML}-types. We capture this situation by that p strictly preserves the bi-CC structure on $\mathbb{P}$, and maps $\dot{T}$ to T (see Definition 2.6). The following two definitions summarises the above discussion.

Definition 5.5 Let Δ be a λ_{ML}-signature. An $\lambda_{\mathrm{ML}}(\Delta)$-*structure with predicates* consists of the following data.

1. A $\lambda_{\mathrm{ML}}(\Delta)$-structure $(\mathbb{C}, \mathsf{T}, [\![-]\!])$.
2. A bi-CCC $\mathbb{P}$. We denote the bi-CC structure with a dot, like $\dot{\times}, \dot{+}, \dot{\Rightarrow}, \dot{\pi}, \cdots$.
3. A faithful functor $p : \mathbb{P} \to \mathbb{C}$ such that p strictly preserves the bi-CC structure, and each $\mathbb{P}_I$ has the largest element $\top_I$.

RIGHTSLINK

The common typing rules for EFe(Σ) and EFi(Σ) (in EFi(Σ) the type annotation x^τ at a variable binder is replaced by $x^{|\tau|}$)

$$\frac{\Gamma \vdash M_i : \tau_i \quad 1 \le i \le n}{\Gamma \vdash \langle M_1, \cdots, M_n \rangle : \prod(\tau_1, \cdots, \tau_n)} \qquad \frac{\Gamma \vdash M : \prod(\tau_1, \cdots, \tau_n) \quad 1 \le i \le n}{\Gamma \vdash \pi_i\, M : \tau_i}$$

$$\frac{\Gamma \vdash M : \tau_i \quad 1 \le i \le n}{\Gamma \vdash \iota_i\, M : \coprod(\tau_1, \cdots, \tau_n)} \qquad \frac{\Gamma \vdash M : \coprod(\tau_1, \cdots, \tau_n) \quad \Gamma, x_i : \tau_i \vdash M_i : \tau \quad 1 \le i \le n}{\Gamma \vdash \mathsf{case}\ M\ \mathsf{with}\ \iota_1\, x_1^{\tau_1}\, .\, M_1, \cdots, \iota_n\, x_n^{\tau_n}\, .\, M_n : \tau}$$

$$\frac{\Gamma, x : \tau \vdash M : \tau'}{\Gamma \vdash \lambda x^\tau\, .\, M : \tau \Rightarrow \tau'} \qquad \frac{\Gamma \vdash M : \tau \Rightarrow \tau' \quad \Gamma \vdash N : \tau}{\Gamma \vdash MN : \tau'}$$

$$\frac{\Gamma \vdash M : \beta' \Rightarrow T(e', \tau) \quad s(o) = H(\beta, \beta', e)}{\Gamma \vdash o(M) : \beta \Rightarrow T(ee', \tau)} \qquad \frac{\Gamma \vdash M_1 : T(e_1, \tau) \quad \cdots \quad \Gamma \vdash M_n : T(e_n, \tau) \quad s(o) = I(n, \epsilon)}{\Gamma \vdash o(M_1, \cdots, M_n) : T(\epsilon(e_1, \cdots, e_n), \tau)} \qquad \frac{}{\Gamma \vdash [M] : T(1, \tau)}$$

Typing rules specific to EFe(Σ)

$$\frac{\Gamma \vdash M : T(e, \tau) \quad e \le e'}{\Gamma \vdash T^{e \lesssim e'}\, M : T(e', \tau)} \qquad \frac{\Gamma \vdash M : T(e, \tau) \quad \Gamma, x : \tau \vdash N : T(e', \tau')}{\Gamma \vdash \mathsf{let}^{e,e'}\ x^\tau\ \mathsf{be}\ M\ \mathsf{in}\ N : T(e \cdot e', \tau')}$$

Typing rules specific to EFi(Σ)

$$\frac{\Gamma \vdash M : T(e, \tau) \quad e \le e'}{\Gamma \vdash M : T(e', \tau)} \qquad \frac{\Gamma \vdash M : T(e, \tau) \quad \Gamma, x : \tau \vdash N : T(e', \tau')}{\Gamma \vdash \mathsf{let}\ x^{|\tau|}\ \mathsf{be}\ M\ \mathsf{in}\ N : T(e \cdot e', \tau')}$$

Figure 1. Typing Rules of EFe(Σ) and EFi(Σ)

$$T^{e \lesssim e}\, M = M \tag{6}$$
$$T^{e' \lesssim e''}\, (T^{e \lesssim e'}\, M) = T^{e \lesssim e''}\, M \tag{7}$$
$$\mathsf{let}^{e',e''}\ x^\tau\ \mathsf{be}\ T^{e \lesssim e'}\, M\ \mathsf{in}\ N = T^{ee'' \lesssim e'e''}(\mathsf{let}^{e,e''}\ x^\tau\ \mathsf{be}\ M\ \mathsf{in}\ N) \tag{8}$$
$$\mathsf{let}^{e'',e'}\ x^\tau\ \mathsf{be}\ M\ \mathsf{in}\ T^{e \lesssim e'}\, N = T^{e''e \lesssim e''e'}(\mathsf{let}^{e'',e}\ x^\tau\ \mathsf{be}\ M\ \mathsf{in}\ N) \tag{9}$$
$$\mathsf{let}^{1,e}\ x^\tau\ \mathsf{be}\ [M]\ \mathsf{in}\ N = N[M/x] \tag{10}$$
$$\mathsf{let}^{e,1}\ x^\tau\ \mathsf{be}\ M\ \mathsf{in}\ [x] = M \tag{11}$$
$$\mathsf{let}^{e,e'e''}\ x^\tau\ \mathsf{be}\ M\ \mathsf{in}\ \mathsf{let}^{e',e''}\ y^\sigma\ \mathsf{be}\ N\ \mathsf{in}\ L = \mathsf{let}^{ee',e''}\ y^\sigma\ \mathsf{be}\ (\mathsf{let}^{e,e'}\ x^\tau\ \mathsf{be}\ M\ \mathsf{in}\ N)\ \mathsf{in}\ L \tag{12}$$
$$\mathsf{let}^{ee',e''}\ x^\tau\ \mathsf{be}\ o(M)\ N\ \mathsf{in}\ L = o(\lambda y^\beta\, .\, \mathsf{let}^{e',e''}\ x^\tau\ \mathsf{be}\ M\ y\ \mathsf{in}\ L)\ N \quad (s(o) = H(\beta, \beta', e)) \tag{13}$$
$$\mathsf{let}^{\epsilon(e_1,\cdots,e_n),e'}\ x^\tau\ \mathsf{be}\ o(M_1, \cdots, M_n)\ \mathsf{in}\ N = o(\mathsf{let}^{e_1,e'}\ x^\tau\ \mathsf{be}\ M_1\ \mathsf{in}\ N, \cdots, \mathsf{let}^{e_n,e'}\ x^\tau\ \mathsf{be}\ M_n\ \mathsf{in}\ N) \quad (s(o) = I(n, \epsilon)) \tag{14}$$

Figure 2. Equational Axioms for EFe(Σ)

The notation for a $\lambda_{\mathrm{ML}}(\Delta)$-structure with predicates is $p : \mathbb{P} \to (\mathbb{C}, \mathsf{T}, [\![-]\!])$.

We note that the largest element $\top_I$ in the fibre $\mathbb{P}_I$ corresponds to I itself in $\mathbb{P}$, or the predicate *true*.

Definition 5.6 An EFi(Σ)-*structure* is a pair of a $\lambda_{\mathrm{ML}}(\Sigma_0)$-structure $p : \mathbb{P} \to (\mathbb{C}, \mathsf{T}, [\![-]\!])$ with predicates and a strong parametric $\mathbb{E}$-monad $\dot{T}$ on $\mathbb{P}$ such that

1. The functor p maps $\dot{T}$ to T (see Definition 2.6).
2. For each $o \in O$ such that $s(o) = H(\beta, \beta', e)$, the $([\![\beta]\!], [\![\beta']\!])$-ary algebraic operation $[\![o]\!]$ for T satisfies: for all $e' \in \mathbb{E}, X \in \mathbb{P}$,
$$[\![o]\!]_{pX} : \top_{[\![\beta']\!]} \dot{\Rightarrow} \dot{T}e'X \dot{\to} \top_{[\![\beta]\!]} \dot{\Rightarrow} \dot{T}(ee')X.$$
3. For each $o \in O$ such that $s(o) = I(n, \epsilon)$, the n-ary algebraic operation $[\![o]\!]$ for T satisfies: for all $e_1, \cdots, e_n \in \mathbb{E}, X \in \mathbb{P}$,
$$[\![o]\!]_{pX} : \dot{T}e_1X \dot{\times} \cdots \dot{\times} \dot{T}e_nX \dot{\to} \dot{T}(\epsilon(e_1, \cdots, e_n))X.$$

The notation for an EFi(Σ)-structure is $p : (\mathbb{P}, \dot{T}) \to (\mathbb{C}, T, [\![-]\!])$.

Given an EFi(Σ)-structure $p : (\mathbb{P}, \dot{T}) \to (\mathbb{C}, \mathsf{T}, [\![-]\!])$, we interpret the calculus EFi(Σ) as follows. We interpret each type $\tau \in \mathrm{Typ}_{\mathrm{EF}}(\Sigma)$ by an object $\langle \tau \rangle \in \mathbb{P}$:

$$\begin{aligned} \langle b \rangle &= \top_{[\![b]\!]} \\ \langle \textstyle\prod(\tau_1, \cdots, \tau_n) \rangle &= \dot{\prod}(\langle \tau_1 \rangle, \cdots, \langle \tau_n \rangle) \\ \langle \textstyle\coprod(\tau_1, \cdots, \tau_n) \rangle &= \dot{\coprod}(\langle \tau_1 \rangle, \cdots, \langle \tau_n \rangle) \\ \langle \tau \Rightarrow \tau' \rangle &= \langle \tau \rangle \dot{\Rightarrow} \langle \tau' \rangle \\ \langle T e \tau \rangle &= \dot{T} e \langle \tau \rangle. \end{aligned}$$

We have $p\langle \tau \rangle = [\![|\tau|]\!]$, thus refinement types are indeed interpreted by a predicate over $[\![|\tau|]\!]$. We then show that the interpretation of the underlying $\lambda_{\mathrm{ML}}(\Sigma_0)$-judgement of an EFi(Σ)-judgement respects the predicates given by refinement types.

Theorem 5.7 *Let $p : (\mathbb{P}, \dot{T}) \to (\mathbb{C}, \mathsf{T}, [\![-]\!])$ be an* EFi(Σ)*-structure. For all* EFi(Σ)*-judgements $x : \tau_1, \cdots, x_n : \tau_n \vdash M : \tau$, the interpretation of the $\lambda_{\mathrm{ML}}(\Sigma_0)$-judgement $x : |\tau_1|, \cdots, x_n : |\tau_n| \vdash M : |\tau|$ (which is derivable by Proposition 5.4) in the $\lambda_{\mathrm{ML}}(\Sigma_0)$-structure $(\mathbb{C}, \mathsf{T}, [\![-]\!])$ satisfies:*
$$[\![M]\!] : \langle \tau_1 \rangle \dot{\times} \cdots \dot{\times} \langle \tau_n \rangle \dot{\to} \langle \tau \rangle.$$

5.3 Soundness of EFi to Effect Specifications

We next discuss the soundness of EFi with respect to *effect specifications*. Recall that the primary purpose of effect system is to statically estimate computational effects caused by programs. To show the soundness of the estimation, we first need to specify the

RIGHTSLINK

meaning, that is, the scope of computational effects, of each effect. Then the soundness of an effect system is that for every program M having an effect e, the computational effect of M is included in the scope assigned to e.

In our semantic framework, this discussion is roughly formulated as follows. Let $p : \mathbb{P} \to (\mathbb{C}, \mathsf{T}, [\![-]\!])$ be a $\lambda_{\mathrm{ML}}(\Sigma_0)$-structure with predicates. We specify the meaning of each effect e as a $\mathbb{P}$-object $S_b e$ above $\mathrm{T}[\![b]\!]$; here b is a base type. We then say that EFi(Σ) is sound with respect to the specification S of effects if for all judgements $M : T(e, b)$ in EFi, $[\![M]\!] \in S_b e$. We actually allow certain type of free variables to occur in M.

Definition 5.8 Let $p : \mathbb{P} \to (\mathbb{C}, \mathsf{T}, [\![-]\!])$ be a $\lambda_{\mathrm{ML}}(\Sigma_0)$-structure with predicates. An *effect specification* of Σ in p is just a family of functors (i.e. monotone mappings) $\{S_b : \mathbb{E} \to \mathbb{P}_{\mathrm{T}[\![b]\!]}\}_{b \in B}$.

Definition 5.9 Let $p : \mathbb{P} \to (\mathbb{C}, \mathsf{T}, [\![-]\!])$ be a $\lambda_{\mathrm{ML}}(\Sigma_0)$-structure with predicates and S be an effect specification of Σ in p. We say that EFi(Σ) is:

1. *rank-0 sound with respect to S* if for all EFi(Σ)-judgements
$$x_1 : b_1, \cdots, x_n : b_n \vdash M : T(e, b),$$
the interpretation $[\![M]\!]$ of its $\lambda_{\mathrm{ML}}(\Sigma_0)$-judgement in $(\mathbb{C}, \mathsf{T}, [\![-]\!])$ satisfies
$$[\![M]\!] : \top_{[\![b_1]\!]} \dot{\times} \cdots \dot{\times} \top_{[\![b_n]\!]} \dot{\to} S_b e.$$
2. *rank-1 sound with respect to S* if for all EFi(Σ)-judgements
$$x_1 : \beta_1 \Rightarrow T(e_1, b_1), \cdots, x_n : \beta_n \Rightarrow T(e_n, b_n) \vdash M : \beta \Rightarrow T(e, b),$$
the interpretation $[\![M]\!]$ of its $\lambda_{\mathrm{ML}}(\Sigma_0)$-judgement satisfies
$$[\![M]\!] : \dot{\prod}_{i=1}^{n} (\top_{[\![\beta_i]\!]} \dot{\Rightarrow} S_{b_i} e_i) \dot{\to} (\top_{[\![\beta]\!]} \dot{\Rightarrow} S_b e).$$

Below, by imposing a mild condition on the faithful functor $p : \mathbb{P} \to \mathbb{C}$ of a λ_{ML}-structure with predicates, we show that the soundness of EFi is derivable from (a combination of) the following closure properties on effect specifications.

Definition 5.10 (Continued from Definition 5.8) We say that:

- S is *closed under algebraic operations in* Σ if it satisfies the following two sub-conditions:
  A) For each $o \in O$ such that $s(o) = H(\beta, \beta', e)$, the $([\![\beta]\!], [\![\beta']\!])$-ary algebraic operation $[\![o]\!]$ for T satisfies: for all $e' \in \mathbb{E}, b \in B$,
  $$[\![o]\!]_{[\![b]\!]} : \top_{[\![\beta']\!]} \dot{\Rightarrow} S_b e' \dot{\to} \top_{[\![\beta]\!]} \dot{\Rightarrow} S_b(ee').$$
  B) For each $o \in O$ such that $s(o) = I(n, \epsilon)$, the n-ary algebraic operation $[\![o]\!]$ for T satisfies: for all $e_1, \cdots, e_n \in \mathbb{E}, b \in B$,
  $$[\![o]\!]_{[\![b]\!]} : \dot{\prod}_{i=1}^{n} S_b e_i \dot{\to} S_b(\epsilon(e_1, \cdots, e_n)).$$
- S *contains values* if for all $b \in B$, $\eta_{[\![b]\!]} : \top_{[\![b]\!]} \dot{\to} S_b 1$.
- S is *closed under lifting* if the unit
  $$\sigma_I^{\mathsf{T},R} : \mathrm{T}I \to (I \Rightarrow \mathrm{T}R) \Rightarrow \mathrm{T}R$$
  of the continuation monad transformer [5] satisfies: for all $e, e' \in \mathbb{E}, b, b' \in B$,
  $$\sigma_{[\![b]\!]}^{\mathsf{T},[\![b']\!]} : S_b e \dot{\to} (\top_{[\![b]\!]} \dot{\Rightarrow} S_{b'} e') \dot{\Rightarrow} S_{b'}(ee').$$

The condition we impose on p is that: p is a *fibration with fibred small products*. Such fibrations are often used in the categorical formulation of logical relations; see [11, 15]. Under this condition, we can *construct* a strong parametric $\mathbb{E}$-monad on $\mathbb{P}$ from effect specifications. It is a variation of the categorical $\top\top$-lifting in [14].

The construction proceeds as follows. Let $p : \mathbb{P} \to (\mathbb{C}, \mathsf{T}, [\![-]\!])$ be a $\lambda_{\mathrm{ML}}(\Sigma_0)$-structure with predicates such that $p : \mathbb{P} \to \mathbb{C}$ is a fibration with fibred small products. Also, let S be an effect specification of Σ in p. For each $X \in \mathbb{P}$, we first define an auxiliary $\mathbb{P}$-object $\dot{T}^{e',b} eX$ $(e, e' \in \mathbb{E}, b \in B)$ by the following inverse image in the fibration p:

$$\begin{array}{ccc} \dot{T}^{e',b} eX & \dashrightarrow & (X \dot{\Rightarrow} S_b e') \dot{\Rightarrow} S_b(ee') \\ \downarrow & & \downarrow p \\ \mathrm{T}(pX) & \xrightarrow[\sigma_{pX}^{\mathsf{T},[\![b]\!]}]{} & (pX \Rightarrow \mathrm{T}[\![b]\!]) \Rightarrow \mathrm{T}[\![b]\!] \end{array} \qquad \begin{array}{c} \mathbb{P} \\ \downarrow p \\ \mathbb{C} \end{array}$$

(see also Example 2.11). We then define $\dot{T}^S eX$ to be the following small product (i.e. meet) in the fibre category $\mathbb{P}_{\mathrm{T}(pX)}$:

$$\dot{T}^S eX = \bigwedge_{e' \in \mathbb{E}, b \in B} \dot{T}^{e',b} eX.$$

Theorem 5.11 *Let $p : \mathbb{P} \to (\mathbb{C}, \mathsf{T}, [\![-]\!])$ be a $\lambda_{\mathrm{ML}}(\Sigma_0)$-structure with predicates such that p is a fibration with fibred small products. Also, let S be an effect specification of Σ in p.*

1. *$\dot{T}^S$ is a strong parametric $\mathbb{E}$-monad on $\mathbb{P}$, and p maps $\dot{T}^S$ to T.*
2. *If S is closed under algebraic operations, then we have an* EFi(Σ)*-structure $p : (\mathbb{P}, \dot{T}^S) \to (\mathbb{C}, \mathsf{T}, [\![-]\!])$.*
3. *If S is closed under algebraic operations and contains values, then* EFi(Σ) *is rank-0 sound with respect to S.*
4. *S is closed under algebraic operations, contains values and is closed under lifting if and only if* EFi(Σ) *is rank-1 sound with respect to S.*

Theorem 5.11-3 is a parametric analogue of [15, Theorem 7].

Example 5.12 Let $\Delta = (B, O, s)$ be a λ_{ML}-signature, $(\mathbf{Set}, \mathsf{T}, [\![-]\!])$ be a $\lambda_{\mathrm{ML}}(\Delta)$-structure and $\alpha : \mathsf{T} \to (\mathsf{S}, \sqsubseteq)$ be an effect observation. We assume that Δ contains only natural number-ary algebraic operations. The subobject fibration $p : \mathbf{Sub}(\mathbf{Set}) \to \mathbf{Set}$ (see[11, Chapter 0]) provides the data 2 and 3 of Definition 5.5, and has fibred small products. Therefore $p : \mathbf{Sub}(\mathbf{Set}) \to (\mathbf{Set}, T, [\![-]\!])$ is a $\lambda_{\mathrm{ML}}(\Delta)$-structure with predicates.

We next derive an EF-signature $\Sigma_\Delta = (\mathsf{S}\mathbf{1}, B, O, s')$ where s' is defined by
$$s'(o) = (n, \overline{\alpha}([\![o]\!])_{\mathbf{1}}) \quad (s(o) = I(n)).$$
This s' assigns the effect function to each operator symbol in O by Theorem 4.3-1.

We give an effect specification S of Σ_Δ in p by means of the parametric effect monad D derived by Theorem 3.3: $S_b e = (De[\![b]\!] \subseteq \mathrm{T}[\![b]\!])$. This is closed under algebraic operations in Σ_Δ thanks to Theorem 4.3-2, and moreover S contains values and is closed under lifting because D is a parametric effect monad. By Theorem 5.11-4, EFi(Σ) is rank-1 sound with respect to S, that is, for every EFi(Σ_Δ)-judgement
$$x_1 : \beta_1 \Rightarrow T(e_1, b_1), \cdots, x_n : \beta_n \Rightarrow T(e_n, b_n) \vdash M : \beta \Rightarrow T(e, b),$$
the interpretation $[\![M]\!]$ of its $\lambda_{\mathrm{ML}}(\Delta)$-judgement in $(\mathbf{Set}, \mathsf{T}, [\![-]\!])$ satisfies: for all functions $f_i : [\![\beta_i]\!] \to De_i[\![b_i]\!]$ and $x \in [\![\beta']\!]$, we have
$$[\![M]\!]\{x_1 : f_1, \cdots, x_n : f_n\}(x) \in De'[\![b']\!].$$

6. Related Work

6.1 Effectors and Productors

Tate introduced a semantic structure called *effectors* and *productors* in [33] as a solution to Problem 1.1 and 1.2. Through a characterisation of effectors / productors in terms of *multicategory theory*, we show that parametric effect monads are *productors on total principalled effectoids*. A good reference for multicategory theory is [20].

Let us write **MultCAT** for the (super-large) category of multicategories and maps between them. There is an adjunction relating

RIGHTSLINK

multicategories and strict monoidal categories [20, Section 2.3]:

$$\mathbf{StrictMonCAT} \underset{F}{\overset{U}{\underset{\longleftarrow}{\overset{\longrightarrow}{\top}}}} \mathbf{MultCAT}$$

The right adjoint U maps a strict monoidal category $(\mathbb{C}, \mathbf{I}, \otimes)$ to its *underlying multicategory* $U\mathbb{C}$ [20, Example 2.1.3]. It has the same objects as $\mathbb{C}$, and its homsets are defined by

$$(U\mathbb{C})([I_1, \cdots, I_n], J) = \mathbb{C}(I_1 \otimes \cdots \otimes I_n, J).$$

The functor U preserves thinness. We also mention a relationship between lax monoidal functors and maps for multicategories, stated as Example 2.1.10 in [20]. For each strict monoidal category $\mathbb{B}$ and $\mathbb{D}$, we have the following bijection:

$$\mathbf{LaxMonCAT}(\mathbb{B}, \mathbb{D}) \simeq \mathbf{MultCAT}(U\mathbb{B}, U\mathbb{D}). \tag{15}$$

We omit the definition of effectors and productors; see [33, Section 5 & 6]. We give the following multicategorical characterisation of effectors and productors:

Theorem 6.1 *1. [33, Section 9] Effectors bijectively correspond to thin small multicategories. (Below we identify these two concepts.)*

2. Let $\mathbb{F}$ be an effector. Then $\mathbb{F}$-productors bijectively correspond to maps (for multicategories) of type $\mathbb{F} \to U[\mathbb{C}, \mathbb{C}]$.

For a preordered monoid $\mathbb{E}$, $U\mathbb{E}$ is an effector called *total principalled effectoid* [33, Section 9]. The following proposition is a consequence of (15):

Proposition 6.2 *Let $\mathbb{E}$ be a preordered monoid. Then parametric $\mathbb{E}$-monads on a category $\mathbb{C}$ bijectively correspond to $U\mathbb{E}$-productors on $\mathbb{C}$.*

We end this section with the discussion about the unification of productors and parametric effect monads. Let $\mathbb{F}$ be an effector. Using the adjunction $F \dashv U$ and the inclusion of categories $\mathbf{StrictMonCAT} \subseteq \mathbf{LaxMonCAT}$, we obtain

$$\begin{aligned} \mathbf{MultiCAT}(\mathbb{F}, U[\mathbb{C}, \mathbb{C}]) &\simeq \mathbf{StrictMonCAT}(F\mathbb{F}, [\mathbb{C}, \mathbb{C}]) \\ &\subseteq \mathbf{LaxMonCAT}(F\mathbb{F}, [\mathbb{C}, \mathbb{C}]). \end{aligned}$$

We note that $F\mathbb{F}$ is not thin in general. This suggests that productors can be encoded by general parametric monads [25].

6.2 Actions of Monoidal Categories

Parametric effect monads are an instance of the broader concept of *action of monoidal categories*. Depending on the degree of the preservation of monoidal structure, there are several variations of this concept, each of which corresponds to *strong*, *lax* and *oplax* monoidal functors of type $\mathbb{M} \to [\mathbb{C}, \mathbb{C}]$.

In mathematics, a category equipped with a strong action of a monoidal category is sometimes called an *actegory* (not a typo!), and studied in e.g. [17]. In Levy's *call-by-push-value*, a strong action of a category with finite products is used as a part of a CBPV value / producer model [21].

Parametric effect monads are exactly lax actions of preordered monoids. Lax actions of general monoidal categories are called *parametric monads* in [25] and *negative $\mathbb{M}$-categories* in [24]. They are introduced to give a unified account of the *tensorial strength* of the continuation monad and the *weak distributive law* in linear logic. The categorical analysis of parametric monads in [24, 25] are readily applicable to parametric effect monads. Especially, the concept of *commutator* for parametric monads in [25] will provide a method to synthesise parametric effect monads, hence effect systems as well. Compared to [24, 25], this paper provides new examples and constructions of parametric (effect) monads arising from the study of a semantics of effect system. We note that parametric effect Kleisli triples and the parametric state / continuation monad can be defined for general monoidal categories.

Recently, Petricek et al. study oplax actions of monoids to categories under the name *indexed comonad* [29, Definition 2]. Although they are the dual of parametric effect monads, the spirit of their work is very close to this work. It is interesting to see whether we can dualise the construction of parametric effect monads using effect observations.

We finally mention a possible generalisation of parametric (effect) monads. A monoidal category $\mathbb{M}$ and a lax monoidal functor of type $\mathbb{M} \to [\mathbb{C}, \mathbb{C}]$ bijectively corresponds to the one-object bicategory $\mathbf{B}\mathbb{M}$ and a lax functor of type $\mathbf{B}\mathbb{M} \to \mathbf{B}[\mathbb{C}, \mathbb{C}]$, respectively. This shift to the bicategory theory suggests us to take a bicategory $\mathbb{E}$ and a lax functor of type $\mathbb{E} \to \mathbf{B}[\mathbb{C}, \mathbb{C}]$ as a generalisation of the monoid structure on effects and the parametric effect monad. An effect is now a *1-cell* $e : X \to Y$ in $\mathbb{E}$, and a parametric effect monad is indexed by the domain and codomain of e, like $T_{X,Y}(e)$. A future work is to examine whether we can extend the techniques developed in this paper to bicategorical parametric monads.

6.3 Parameterised Moands

Atkey's parameterised monads [2] are another generalisation of monads. They seem to be fundamentally different from parametric effect monads. To exhibit this difference, we recall that one-object $[\mathbb{C}, \mathbb{C}]$-enriched categories are monads on $\mathbb{C}$. Thus $[\mathbb{C}, \mathbb{C}]$-enriched categories are a generalisation of monads; they correspond to *discretely parameterised monads*. Parameterised monads are a further generalisation of such enriched categories. Tate shows an encoding of preorder-parameterised monads as productors [33, Section 7]. At this moment we do not know if it is possible to directly encode parameterised monads as parametric effect monads.

6.4 Denotational Semantics of Effect Systems

The implicit subeffecting system and its denotational semantics are used in the study of effect-dependent program optimisation [4, 6–8]. The semantic framework we introduced in Definition 5.5 and 5.6 is influenced by these works, which use PER-like categories as the categories of predicates. They manually construct the denotation of effect-annotated monadic types on the category of predicates, while in this paper we give a mechanical method to construct parametric effect monads by a variant of the semantic $\top\top$-lifting.

7. Conclusion

Under the formulation of preordered monoids as a structure on effects and parametric effect monads as a semantic structure for the effect-annotated monadic type, we studied their properties, constructions and applications to the semantics of effect systems. Parametric effect monads admit various analogues of the structures that exist in the theory of monad. We gave a construction of parametric effect monads from effect observations, and semantics of implicit subeffecting calculus EFi. We then discussed the soundness of EFi with respect to a given specification of effects.

A future work is to extend the arity of Definition 4.2 to arbitrary objects so that we can handle fresh name creations and local stores. It is also interesting to extend the construction in Theorem 3.3 to the one over a general category $\mathbb{C}$.

Acknowledgment

Thanks to Masahito Hasegawa, Susumu Nishimura, Paul-André Melliès, Marco Gaboardi and anonymous reviewers for their helpful comments and suggestions on this paper. Additionally, thanks to Kohei Suenaga for an insightful discussion in Tanakaya. This work was supported by JSPS KAKENHI Grant Number 24700012.

RIGHTSLINK

References

[1] R. M. Amadio and P.-L. Curien. *Domains and lambda-calculi*. Cambridge University Press, New York, NY, USA, 1998.

[2] R. Atkey. Parameterised notions of computation. *J. Funct. Program.*, 19(3-4):335–376, 2009.

[3] G. Barthe, P. Dybjer, L. Pinto, and J. Saraiva, editors. *Applied Semantics, International Summer School, APPSEM 2000, Caminha, Portugal, September 9-15, 2000, Advanced Lectures*, volume 2395 of *LNCS*. Springer, 2002.

[4] N. Benton and P. Buchlovsky. Semantics of an effect analysis for exceptions. In *Proceedings of 2007 ACM SIGPLAN International Workshop on Types in Languages Design and Implementation*, pages 15–26. ACM, 2007.

[5] N. Benton, J. Hughes, and E. Moggi. Monads and effects. In Barthe et al. [3], pages 42–122.

[6] N. Benton, A. Kennedy, L. Beringer, and M. Hofmann. Relational semantics for effect-based program transformations with dynamic allocation. In *Proceedings of the 9th International ACM SIGPLAN Conference on Principles and Practice of Declarative Programming*, pages 87–96. ACM, 2007.

[7] N. Benton, A. Kennedy, L. Beringer, and M. Hofmann. Relational semantics for effect-based program transformations: higher-order store. In *Proceedings of the 11th International ACM SIGPLAN Conference on Principles and Practice of Declarative Programming*, pages 301–312. ACM, 2009.

[8] N. Benton, A. Kennedy, M. Hofmann, and L. Beringer. Reading, writing and relations. In *Programming Languages and Systems, 4th Asian Symposium, APLAS 2006, Proceedings*, volume 4279 of *LNCS*, pages 114–130. Springer, 2006.

[9] A. Filinski. Representing layered monads. In *Proc. POPL 1999*, pages 175–188, 1999.

[10] J.-C. Filliâtre. A theory of monads parameterized by effects. Research Report 1367, LRI, Université Paris Sud, November 1999.

[11] B. Jacobs. *Categorical Logic and Type Theory*. Elsevier, 1999.

[12] B. Jacobs. Coalgebraic trace semantics for combined possibilistic and probabilistic systems. *Electr. Notes Theor. Comput. Sci.*, 203(5):131–152, 2008.

[13] O. Kammar and G. D. Plotkin. Algebraic foundations for effect-dependent optimisations. In *Proceedings of the 39th ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages*, pages 349–360. ACM, 2012.

[14] S. Katsumata. A semantic formulation of TT-lifting and logical predicates for computational metalanguage. In *Proc. CSL '05*, volume 3634 of *LNCS*, pages 87–102. Springer, 2005.

[15] S. Katsumata. Relating computational effects by TT-lifting. *Inf. Comput.*, 222:228–246, 2013.

[16] S. Katsumata and T. Sato. Preorders on monads and coalgebraic simulations. In *Foundations of Software Science and Computation Structures - 16th International Conference, Proceedings*, volume 7794 of *LNCS*, pages 145–160. Springer, 2013.

[17] G.M. Kelly and G. Janelidze. A note on actions of a monoidal category. *Theory and Applications of Categories*, 9(4):61–91, 2001.

[18] A. Kock. Strong functors and monoidal monads. *Archiv der Mathematik*, 23(1):113–120, 1972.

[19] S. Lack and R. Street. The formal theory of monads ii. *Journal of Pure and Applied Algebra*, 175(1-3):243 – 265, 2002.

[20] T. Leinster. *Higher Operads, Higher Categories*. London Mathematical Society Lecture Note Series. Cambridge University Press, 2004.

[21] P. B. Levy. *Call-by-Push-Value A Functiona/Imperative Synthesis*. Springer, 2004.

[22] J. M. Lucassen and D. K. Gifford. Polymorphic effect systems. In *Proceedings of the 15th ACM SIGPLAN-SIGACT symposium on Principles of programming languages*, pages 47–57. ACM, 1988.

[23] S. MacLane. *Categories for the Working Mathematician (Second Edition)*, volume 5 of *Graduate Texts in Mathematics*. Springer, 1998.

[24] P.-A. Mellies. Parametric monads and enriched adjunctions. Manuscript.

[25] P.-A. Mellies. The parametric continuation monad. *Mathematical Structures in Computer Science*, Festschrift in honor of Corrado Böhm for his 90th birthday, 2013.

[26] E. Moggi. Notions of computation and monads. *Inf. Comput.*, 93(1):55–92, 1991.

[27] F. Nielson and H. R. Nielson. From CML to its process algebra. *Theor. Comput. Sci.*, 155:179–219, February 1996.

[28] F. Nielson, H. R. Nielson, and C. Hankin. *Principles of program analysis (2. corr. print)*. Springer, 2005.

[29] T. Petricek, D. A. Orchard, and A. Mycroft. Coeffects: Unified static analysis of context-dependence. In *Automata, Languages, and Programming - 40th International Colloquium, Proceedings*, volume 7966 of *LNCS*, pages 385–397. Springer, 2013.

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

[31] L. Solberg, H. R. Nielson, and F. Nielson. Strictness and totality analysis. In *Static Analysis*, volume 864 of *LNCS*, pages 408–422. Springer Berlin Heidelberg, 1994.

[32] J.-P. Talpin and P. Jouvelot. The type and effect discipline. *Inf. Comput.*, 111(2):245–296, 1994.

[33] R. Tate. The sequential semantics of producer effect systems. In *The 40th Annual ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages*, pages 15–26. ACM, 2013.

[34] J. Thamsborg and L. Birkedal. A kripke logical relation for effect-based program transformations. In *Proceeding of the 16th ACM SIGPLAN International Conference on Functional Programming*, pages 445–456. ACM, 2011.

[35] M. Tofte and J.-P. Talpin. Implementation of the typed call-by-value lambda-calculus using a stack of regions. In *Proceedings of the 21st ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages*, pages 188–201, 1994.

[36] D. Varacca and G. Winskel. Distributing probability over non-determinism. *Mathematical Structures in Computer Science*, 16(1):87–113, 2006.

[37] P. Wadler. The marriage of effects and monads. In *Proceedings of the third ACM SIGPLAN International Conference on Functional Programming (ICFP '98)*, pages 63–74. ACM, 1998.

[38] P. Wadler and P. Thiemann. The marriage of effects and monads. *ACM Trans. Comput. Log.*, 4(1):1–32, 2003.

RIGHTSLINK